NOAA Technical Memorandum NWS WR-201

AN INEXPENSIVE SOLUTION FOR THE MASS DISTRIBUTION OF SATELLITE IMAGES

Glen W. Sampson
George Clark

Salt Lake City, Utah
September 1987

U.S. DEPARTMENT OF COMMERCE / National Oceanic and Atmospheric Administration / National Weather Service

NOAA TECHNICAL MEMORANDA
National Weather Service, Western Region Subseries

The National Weather Service (NWS) Western Region (WR) Subseries provides an informal medium for the documentation and quick dissemination of results not appropriate, or not yet ready, for formal publication. The series is used to report on work in progress, to describe technical procedures and practices, or to relate progress to a limited audience. These Technical Memoranda will report on investigations devoted primarily to regional and local problems of interest mainly to personnel, and hence will not be widely distributed.

Papers 1 to 25 are in the former series, ESSA Technical Memoranda, Western Region Technical Memoranda (WRTM); papers 24 to 59 are in the former series, ESSA Technical Memoranda, Weather Bureau Technical Memoranda (WBTM). Beginning with 60, the papers are part of the series, NOAA Technical Memoranda NWS. Out-of-print memoranda are not listed.

Papers 2 to 22, except for 5 (revised edition), are available from the National Weather Service Western Region, Scientific Services Division, P.O. Box 11188, Federal Building, 125 South State Street, Salt Lake City, Utah 84147. Paper 5 (revised edition), and all others beginning with 25 are available from the National Technical Information Service, U.S. Department of Commerce, Sills Building, 5285 Port Royal Road, Springfield, Virginia 22161. Prices vary for all paper copies; $3.50 microfiche. Order by accession number shown in parentheses at end of each entry.

ESSA Technical Memoranda (WRTM)

2 Climatological Precipitation Probabilities. Compiled by Lucianne Miller, December 1965.
3 Western Region Pre- and Post-FP-3 Program, December 1, 1965, to February 20, 1966. Edward D. Diemer, March 1966.
5 Station Descriptions of Local Effects on Synoptic Weather Patterns. Philip Williams, Jr., April 1966 (revised November 1967, October 1969). (PB-17800)
8 Interpreting the RAREP. Herbert P. Benner, May 1966 (revised January 1967).
11 Some Electrical Processes in the Atmosphere. J. Latham, June 1966.
17 A Digitalized Summary of Radar Echoes within 100 Miles of Sacramento, California. J. A. Youngberg and L. B. Overaas, December 1966.
21 An Objective Aid for Forecasting the End of East Winds in the Columbia Gorge, July through October. D. John Coparanis, April 1967.
22 Derivation of Radar Horizons in Mountainous Terrain. Roger G. Pappas, April 1967.

ESSA Technical Memoranda, Weather Bureau Technical Memoranda (WBTM)

25 Verification of Operation Probability of Precipitation Forecasts, April 1966-March 1967. W. W. Dickey, October 1967. (PB-176240)
26 A Study of Winds in the Lake Mead Recreation Area. R. P. Augulis, January 1968. (PB-177830)
28 Weather Extremes. R. J. Schmidli, April 1968 (revised March 1986). (PB86 177672/AS)
29 Small-Scale Analysis and Prediction. Philip Williams, Jr., May 1968. (PB178425)
30 Numerical Weather Prediction and Synoptic Meteorology. CPT Thomas D. Murphy, USAF, May 1968. (AD 673365)
31 Precipitation Detection Probabilities by Salt Lake ARTC Radars. Robert K. Belesky, July 1968. (PB 179084)
32 Probability Forecasting—A Problem Analysis with Reference to the Portland Fire Weather District. Harold S. Ayer, July 1968. (PB 179289)
36 Temperature Trends in Sacramento—Another Heat Island. Anthony D. Lentini, February 1969. (PB 183055)
37 Disposal of Logging Residues Without Damage to Air Quality. Owen P. Cramer, March 1969. (PB 183057)
39 Upper-Air Lows Over Northwestern United States. A.L. Jacobson, April 1969. PB 184296)
40 The Man-Machine Mix in Applied Weather Forecasting in the 1970s. L.W. Snellman, August 1969. (PB 185068)
43 Forecasting Maximum Temperatures at Helena, Montana. David E. Olsen, October 1969. (PB 185762)
44 Estimated Return Periods for Short-Duration Precipitation in Arizona. Paul C. Kangieser, October 1969. (PB 187763)
46 Applications of the Net Radiometer to Short-Range Fog and Stratus Forecasting at Eugene, Oregon. L. Yee and E. Bates, December 1969. (PB 190476)
47 Statistical Analysis as a Flood Routing Tool. Robert J.C. Burnash, December 1969. (PB 188744)
48 Tsunami. Richard P. Augulis, February 1970. (PB 190157)
49 Predicting Precipitation Type. Robert J.C. Burnash and Floyd E. Hug, March 1970. (PB 190962)
50 Statistical Report on Aeroallergens (Pollens and Molds) Fort Huachuca, Arizona, 1969. Wayne S. Johnson, April 1970. (PB 191743)
51 Western Region Sea State and Surf forecaster's Manual. Gordon C. Shields and Gerald B. Burdwell, July 1970. (PB 193102)
52 Sacramento Weather Radar Climatology. R.G. Pappas and C. M. Veliquette, July 1970. (PB 193347)
54 A Refinement of the Vorticity Field to Delineate Areas of Significant Precipitation. Barry B. Aronovitch, August 1970.
55 Application of the SSARR Model to a Basin without Discharge Record. Vail Schermerhorn and Donal W. Kuehl, August 1970. (PB 194394)
56 Areal Coverage of Precipitation in Northwestern Utah. Philip Williams, Jr., and Werner J. Heck, September 1970. (PB 194389)
57 Preliminary Report on Agricultural Field Burning vs. Atmospheric Visibility in the Willamette Valley of Oregon. Earl M. Bates and David O. Chilcote, September 1970. (PB 194710)
58 Air Pollution by Jet Aircraft at Seattle-Tacoma Airport. Wallace R. Donaldson, October 1970. (COM 71 00017)
59 Application of PE Model Forecast Parameters to Local-Area Forecasting. Leonard W. Snellman, October 1970. (COM 71 00016)

NOAA Technical Memoranda (NWS WR)

60 An Aid for Forecasting the Minimum Temperature at Medford, Oregon, Arthur W. Fritz, October 1970. (COM 71 00120)
63 700-mb Warm Air Advection as a Forecasting Tool for Montana and Northern Idaho. Norris E. Woerner, February 1971. (COM 71 00349)
64 Wind and Weather Regimes at Great Falls, Montana. Warren B. Price, March 1971.
66 A Preliminary Report on Correlation of ARTCC Radar Echoes and Precipitation. Wilbur K. Hall, June 1971. (COM 71 00829)
69 National Weather Service Support to Soaring Activities. Ellis Burton, August 1971. (COM 71 00956)
71 Western Region Synoptic Analysis-Problems and Methods. Philip Williams, Jr., February 1972. (COM 72 10433)
74 Thunderstorms and Hail Days Probabilities in Nevada. Clarence M. Sakamoto, April 1972. (COM 72 10554)
75 A Study of the Low Level Jet Stream of the San Joaquin Valley. Ronald A. Willis and Philip Williams, Jr., May 1972. (COM 72 10707)
76 Monthly Climatological charts of the Behavior of Fog and Low Stratus at Los Angeles International Airport. Donald M. Gales, July 1972. (COM 72 11140)
77 A Study of Radar Echo Distribution in Arizona During July and August. John E. Hales, Jr., July 1972. (COM 72 11136)
78 Forecasting Precipitation at Bakersfield, California, Using Pressure Gradient Vectors. Earl T. Riddiough, July 1972. (COM 72 11146)
79 Climate of Stockton, California. Robert C. Nelson, July 1972. (COM 72 10920)
80 Estimation of Number of Days Above or Below Selected Temperatures. Clarence M. Sakamoto, October 1972. (COM 72 10021)
81 An Aid for Forecasting Summer Maximum Temperatures at Seattle, Washington. Edgar G. Johnson, November 1972. (COM 73 10150)
82 Flash Flood Forecasting and Warning Program in the Western Region. Philip Williams, Jr., Chester L. Glenn, and Roland L. Raetz, December 1972, (revised March 1978). (COM 73 10251)
83 A comparison of Manual and Semiautomatic Methods of Digitizing Analog Wind Records. Glenn E. Rasch, March 1973. (COM 73 10669)
86 Conditional Probabilities for Sequences of Wet Days at Phoenix, Arizona. Paul C. Kangieser, June 1973. (COM 73 11264)
87 A Refinement of the Use of K-Values in Forecasting Thunderstorms in Washington and Oregon. Robert Y.G. Lee, June 1973. (COM 73 11276)
89 Objective Forecast Precipitation Over the Western Region of the United States. Julia N. Paegle and Larry P. Kierulff, September 1973. (COM 73 11946/3AS)
91 Arizona "Eddy" Tornadoes. Robert S. Ingram, October 1973. (COM 73 10465)
92 Smoke Management in the Willamette Valley. EArl M. Bates, May 1974. (COM 74 11277/AS)
93 An Operational Evaluation of 500-mb Type Regression Equations. Alexander E. MacDonald, June 1974. (COM 74 11407/AS)
94 Conditional Probability of Visibility Less than One-Half Mile in Radiation Fog at Fresno, California. John D. Thomas, August 1974. (COM 74 11555/AS)
95 Climate of Flagstaff, Arizona. Paul W. Sorenson, and updated by Reginald W. Preston, January 1987. (PB87 143160/AS)
96 Map type Precipitation Probabilities for the Western Region. Glenn E. Rasch and Alexander E. MacDonald, February 1975. (COM 75 10428/AS)
97 Eastern Pacific Cut-Off Low of April 21-28, 1974. William J. Alder and George R. Miller, January 1976. (PB 250 711/AS)
98 Study on a Significant Precipitation Episode in Western United States. Ira S. Brenner, April 1976. (COM 75 10719/AS)
99 A Study of Flash Flood Susceptibility-A Basin in Southern Arizona. Gerald Williams, August 1975. (COM 75 11360/AS)
102 A Set of Rules for Forecasting Temperatures in Napa and Sonoma Counties. Wesley L. Tuft, October 1975. (PB 246 902/AS)
103 Application of the National Weather Service Flash-Flood Program in the Western Region. Gerald Williams, January 1976. (PB 253 053/AS)
104 Objective Aids for Forecasting Minimum Temperatures at Reno, Nevada, During the Summer Months. Christopher D. Hill, January 1976. (PB 252 866/AS)
105 Forecasting the Mono Wind. Charles P. Ruscha, Jr., February 1976. (PB 254 650)
106 Use of MOS Forecast Parameters in Temperature Forecasting. John C. Plankinton, Jr., March 1976. (PB 254 649)
107 Map Types as Aids in Using MOS PoPs in Western United States. Ira S. Brenner, August 1976. (PB 259 594)
108 Other Kinds of Wind Shear. Christopher D. Hill, August 1976. (PB 260 437/AS)
109 Forecasting North Winds in the Upper Sacramento Valley and Adjoining Forests. Christopher E. Fontana, September 1976. (PB 273 677/AS)
110 Cool Inflow as a Weakening Influence on Eastern Pacific Tropical Cyclones. William J. Denney, November 1976. (PB 264 655/AS)
112 The MAN/MOS Program. Alexander E. MacDonald, February 1977. (PB 265 941/AS)
113 Winter Season Minimum Temperature Formula for Bakersfield, California, Using Multiple Regression. Michael J. Oard, February 1977. (PB 273 694/AS)
114 Tropical Cyclone Kathleen. James R. Fors, February 1977. (PB 273 676/AS)
116 A Study of Wind Gusts on Lake Mead. Bradley Colman, April 1977. (PB 268 847)
117 The Relative Frequency of Cumulonimbus Clouds at the Nevada Test Site as a Function of K-Value. R.F. Quiring, April 1977. (PB 272 831)
118 Moisture Distribution Modification by Upward Vertical Motion. Ira S. Brenner, April 1977. (PB 268 740)
119 Relative Frequency of Occurrence of Warm Season Echo Activity as a Function of Stability Indices Computed from the Yucca Flat, Nevada, Rawinsonde. Darryl Randerson, June 1977. (PB 271 290/AS)
121 Climatological Prediction of Cumulonimbus Clouds in the Vicinity of the Yucca Flat Weather Station. R.F. Quiring, June 1977. (PB 271 704/AS)
122 A Method for Transforming Temperature Distribution to Normality. Morris S. Webb, Jr., June 1977. (PB 271 742/AS)
124 Statistical Guidance for Prediction of Eastern North Pacific Tropical Cyclone Motion - Part I. Charles J. Neumann and Preston W. Leftwich, August 1977. (PB 272 661)
125 Statistical Guidance on the Prediction of Eastern North Pacific Tropical Cyclone Motion - Part II. Preston W. Leftwich and Charles J. Neumann, August 1977. (PB 273 155/AS)
127 Development of a Probability Equation for Winter-Type Precipitation Patterns in Great FAlls, Montana. Kenneth B. Mielke, February 1978. (PB 281 387/AS)
128 Hand Calculator Program to Compute Parcel Thermal Dynamics. Dan Gudgel, April 1978. (PB 283 080/AS)
129 Fire whirls. David W. Goens, May 1978. (PB 283 866/AS)
130 Flash-Flood Procedure. Ralph C. Hatch and Gerald Williams, May 1978. (PB 286 014/AS)
131 Automated Fire-Weather Forecasts. Mark A. Mollner and David E. Olsen, September 1978. (PB 289 916/AS)
132 Estimates of the Effects of Terrain Blocking on the Los Angeles WSR-74C Weather Radar. R.G. Pappas, R.Y. Lee, B.W. Finke, October 1978. (PB 289767/AS)
133 Spectral Techniques in Ocean Wave Forecasting. John A. Jannuzzi, October 1978. (PB291317/AS)
134 Solar Radiation. John A. Jannuzzi, November 1978. (PB291195/AS)
135 Application of a Spectrum Analyzer in Forecasting Ocean Swell in Southern California Coastal Waters. Lawrence P. Kierulff, January 1979. (PB292716/AS)
136 Basic Hydrologic Principles. Thomas L. Dietrich, January 1979. (PB292247/AS)
137 LFM 24-Hour Prediction of Eastern Pacific Cyclones Refined by Satellite Images. John R. Zimmerman and Charles P. Ruscha, Jr., January 1979. (PB294324/AS)
138 A Simple Analysis/Diagnosis System for Real Time Evaluation of Vertical Motion. Scott Heflick and James R. Fors, February 1979. (PB294216/AS)

NOAA Technical Memorandum NWS WR-201

AN INEXPENSIVE SOLUTION FOR THE MASS DISTRIBUTION OF SATELLITE IMAGES

Glen W. Sampson
Scientific Services Division

George Clark
Engineering Division

National Weather Service Western Region
Salt Lake City, Utah
September 1987

UNITED STATES
DEPARTMENT OF COMMERCE
S. Bruce Smart
Secretary
Acting

National Oceanic and
Atmospheric Administration
J. Curtis Mack
Administrator
Acting

National Weather
Service
Richard E. Hallgren, Director

This publication has been reviewed
and is approved for publication by
Scientific Services Division,
Western Region.

Glenn E. Rasch, Chief
Scientific Services Division
Western Region Headquarters
Salt Lake City, Utah

TABLE OF CONTENTS

AN INEXPENSIVE SOLUTION FOR THE MASS DISTRIBUTION OF SATELLITE IMAGES

ABSTRACT

A method for the mass distribution of satellite images is described in detail. This method involves the establishment of a central distribution point which can service numerous remote users via dial telephone connections. The system design is centered around the IBM Personal Computer with costs maintained as low as possible.

The central distribution point receives data from a GOESTAP line, or a WEFAX signal, compresses the data into viable satellite images and services remote user requests. The remote users dial into the central distribution point and retrieve the data. Once data are retrieved, the remote locations have the ability to display single images, animate a series of images, and manipulate the enhancement curve used to display the image(s).

Complete instructions for system duplication are included.

AN INEXPENSIVE SOLUTION FOR THE MASS DISTRIBUTION OF SATELLITE IMAGES

I. INTRODUCTION

Delivering real-time satellite data to National Weather Service Offices (WSOs) has, for a long time, been desirable in the Western Region but prohibitive due to the large costs involved. Numerous solutions and alternatives have been discussed since 1983, although each one involved large per-site costs, which would have made widespread implementation impossible.

Several persistent problems became apparent when different solutions were discussed. These problems were: 1) a need to minimize the cost of the graphic display systems located at the field offices, 2) a need to minimize the communication costs, which translates directly into a requirement for minimal transmission time, and 3) overcoming the technical difficulties in obtaining digital satellite data from existing GOESTAP telephone lines.

Developments over the last 5 years in Personal Computer based graphic display systems have greatly diminished the costs involved in the first problem mentioned above. Thus, targeting an IBM PC as the field site graphic system is the direction this project has taken.

Resolving the second problem (communication costs) requires an approach different from simply allowing the advance of technology to arrive at a solution. The existing Federal Telecommunication System (FTS) does not lend itself to quality data transmissions at rates faster than 1200 baud. To transmit a relatively low resolution digital satellite picture at 1200 baud requires about 15 minutes. This transmission time is unacceptable. Using a higher baud rate, however, limits the telecommunication alternatives available to a field site. These circumstances resulted in the development of a dissemination method using data compression techniques and a 1200 baud dial-up connection, for an average transmission time of about 6 minutes per image.

The third problem is somewhat difficult to overcome and requires hardware which is not in the mainstream of most computer or electronics manufacturers. Therefore, the approach for arriving at a solution has been to use common off-the-shelf hardware wherever possible and develop the remaining hardware only when no other reasonable alternatives existed.

Applying these solutions to the overall problem of getting real-time satellite data to WSO field offices has produced the system described in this document. Efforts were made in this system design to minimize all costs by using common hardware, which may already exist at field sites, and using the full capabilities of any additional hardware which was procured. Normally, when limiting

the cost of a system, the quality of the end product is reduced; we have tried not to fall into this trap and believe we have succeeded.

II. GENERAL SYSTEM DESIGN

Three modular system components have been developed for the WSO Satellite Project (Figure 1). Functionally these system components handle the collection and digitization of data from a GOESTAP line (Analog to Digital Converter), the dissemination of data to the field sites (Dissemination Computer), and the display and manipulation of these data at the field sites (Remote Sites). All of the system modules are independent of each other and communicate only through an asynchronous RS-232 connection. A data flow diagram is given in Figure 1 and depicts the individual system modules. General information on the individual modules follows. Detailed information on the software is found in Chapter 4, Software Documentation, and on the hardware in Chapter 5, Hardware Documentation.

A. The Analog to Digital (A/D) Converter Module

Analog satellite data on a GOESTAP line must be converted to a digital format before any manipulation of the data by a computer can occur. The data flow in this module goes from the GOESTAP line through a signal demodulator into an IBM PC. The IBM PC performs the actual analog to digital conversion, compresses the satellite data into a product for dissemination, and asynchronously transmits the product to the dissemination computer. This PC has the ability to window an image being received so that the full data resolution can be viewed, although normally the data is filtered down to a 640X400 matrix from the full resolution of 2100X459.

The signal demodulator removes the carrier frequency of the waveform, and outputs an analog signal containing only the modulated portion of the data stream. The demodulator is the only portion of the hardware which was developed exclusively for this project, and is not an off-the-shelf component. Complete instructions for assembling this device are found in Chapter 5, Hardware Documentation.

A question which frequently arises concerning the system design is why not have the A/D converter module also handle the dissemination tasks? The amount of data flowing through the A/D converter module in a 24 hour period is over 520 million bytes. This data volume requires the complete computing resources of the PC. In comparison, the data volume of the Automation of Field Operations and Services (AFOS) system at 100% saturation is only 26 million bytes of data in a day.

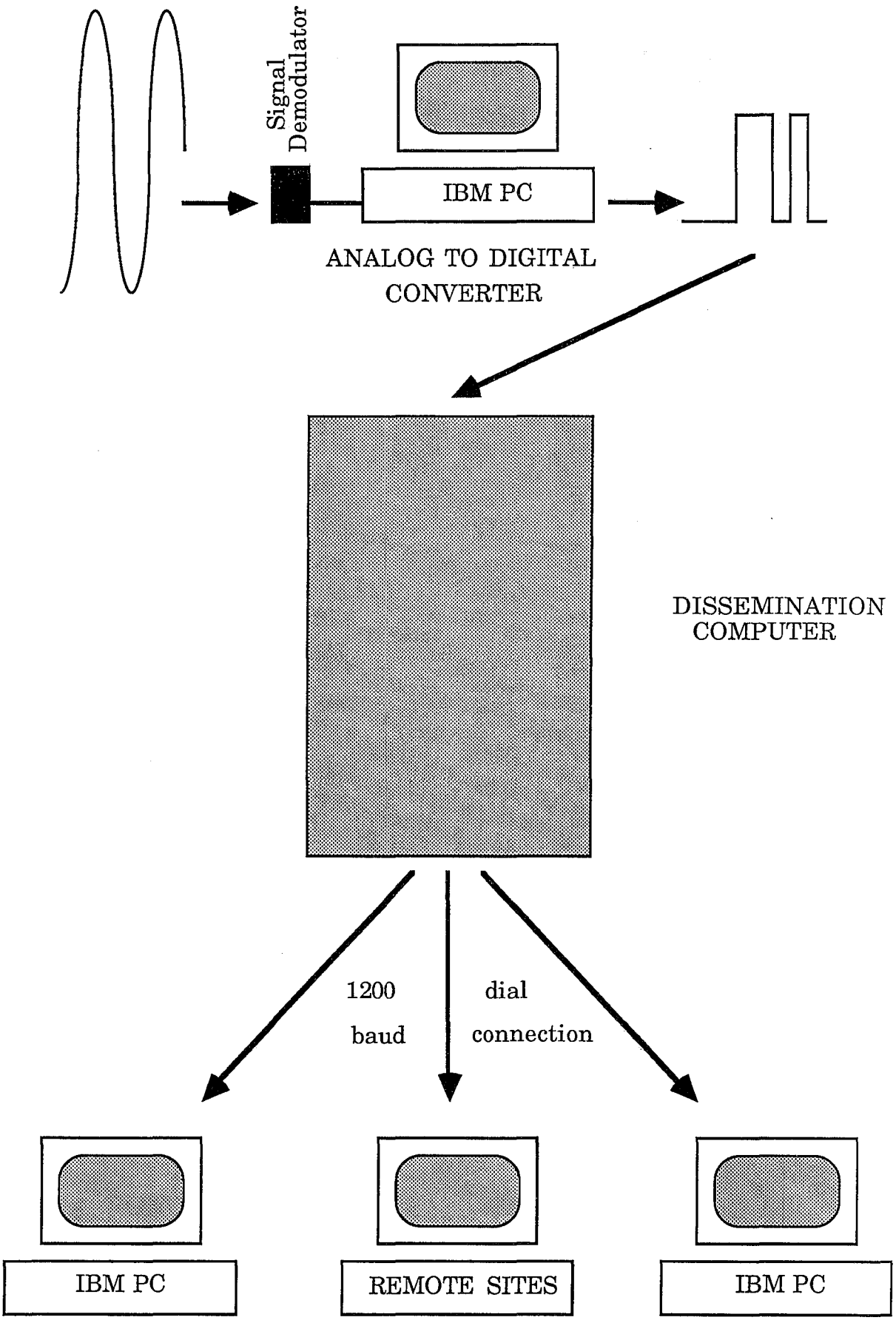


FIGURE 1 - Satellite display system data flow diagram.

B. The Dissemination Computer Module

Functionally, the dissemination computer must receive products from the A/D converter, maintain a simple data base containing these products, and service field sites via dial-up 1200 baud connections. The actual computer system used to accomplish these tasks is directly determined by the number of field sites that will be using the system and their data volume requirements. Numerous types of computers can be used for the dissemination system, anything from another Personal Computer to a timeshare mainframe.

The system that Western Region is currently using is a Data General Eclipse S230 running an AOS operating system. The selection of this dissemination computer system was due to its current availability and existing 1200 baud dial-up ports. No additional procurement costs were incurred to use this system, and software development time was minimal.

C. The Remote Display System

The remote display system must have the capability to initiate a 1200 baud dial-up connection and the necessary hardware to display a low resolution satellite picture. The IBM PC is the computer system used for this module, largely because of its proliferation throughout the NWS, though any computer having these two capabilities can be used if the appropriate software is developed.

The remote display system was designed to use existing IBM hardware wherever possible. All existing IBM PCs in the field can be upgraded to retrieve and display satellite images for less than $1000 each.

Providing communication capabilities on an IBM PC can be accomplished with one of the numerous off-the-shelf packages (e.g. Smartcom or ASCOM). The display portion of this module is provided by using a standard IBM color display and a Tecmar Graphics Master card. The display resolution is 640 x 400 x 16. Other graphics cards can be used (e.g. the IBM EGA), but the Graphics Master provides the desired capabilities using a standard color monitor.

Software has been developed for the display of satellite pictures using this hardware. The software is completely menu driven and provides the capability to display a single image, animate a series of images, and change the enhancement curve used in displaying the image(s). Animation of approximately 3 pictures per second is possible, and a maximum of 9 pictures can be animated (dependent upon the existing memory in the PC). Picture quality is better than NAFAX and less than images received via SWIS.

III. USER'S GUIDE

Using the WSO Satellite Data System consists of executing two sets of programs. First, a program must be used to get the data, and second, a program must be used to display and manipulate the data. The communication program described in this guide is called SATDATA, which controls Hayes or Hayes compatible modems. The display program was specially developed for the hardware described in Chapter 2 and is called SATDSP (short for SATellite DiSPlay). Procedures for using other modems and associated communication software are described in Appendix B.

NOTE: Throughout these instructions **BOLD** entries are responses you would type in on the keyboard, with the <cr> symbol representing the Enter key. CAPITALIZED portions represent the computer response or prompt.

A. Getting the Data

Before the SATDATA program can be executed, a program must be run to gather information about your IBM system. This program is called SATINFO. SATINFO is completely menu driven, so answer the questions as they appear on the screen. SATINFO can be started by typing **satinfo<cr>**. A sample editing session would look like:

```
A>satinfo<cr>

(The screen is cleared.)

PLEASE ANSWER THE QUESTIONS AS THEY APPEAR.
  ALL ENTRIES MUST BE LESS THAN 40 CHARACTERS

IS YOUR HAYES MODEM SETUP AS COM1 (Y/N)? y<cr>
ENTER THE TELEPHONE TO DIAL
INCLUDE COMMAS TO PAUSE (E.G. 8,8015245131)
-> _________________
ENTER YOUR USERNAME -> ___________
ENTER YOUR PASSWORD -> ___________
INFORMATION IS NOW IN THE DISK FILE SATLOGON

A>
```

Obtaining the satellite data requires the user to dial-up the WRH AOS computer system, log onto the system, make a selection from the menu displayed, and log off the system. If you have already run the SATINFO program, many of these steps will be handled for you by the SATDATA program. The following instructions will retrieve current satellite data:

1. Assuming that your IBM is already up and running, insert the Satellite Software diskette, and type **satdata<cr>**. The computer response should be:

   IS THE TELEPHONE LINE READY FOR USE (Y/N)? **y<cr>**

   Answer with a "y" to get:

   EXECUTING THE AUTODIAL
   DIALING (telephone number)

   If a connection is not established with the WRH AOS computer, SATDATA will try indefinitely to establish one. This infinite dialing can be aborted by striking the **ESC** key.

2. Once a connection is established the following is displayed:

   (The screen is cleared.)
   FUNCTION KEYS ARE AS FOLLOWS:
     F1 - CAPTURE DATA AND STORE IT ON DISK
     F10 - RETURN TO DOS (EXIT AOS FIRST)
     ALL OTHER FUNCTION KEYS ARE DISABLED
   YOU ARE CURRENTLY ON LINE TO AOS
   EXECUTING AUTOLOGON

   ---WELCOME TO SATELLITE DATA ACCESS---

   MENU SELECTIONS ARE:
        1. LIST AND SELECT AVAILABLE PICTURES
        2. EXPLANATION OF THE PICTURE TITLES
        3. EXIT
   PLEASE ENTER THE NUMBER OF YOUR SELECTION ->

   Option 1 gives you a list of the current pictures available for you to retrieve. Option 2 provides a description of the picture titles displayed in option 1, and option 3 logs you off of the system.

   The general format of the picture titles is hhmm_DDMMYY_HH_SSs. Where hhmm is the time of the picture, DDMMYY is the day, month and year respectively, HH is the enhancement curve and SSs is the sector information concerning the picture. The enhancement curve (HH) is blank for visual images. Appendix D contains a sample listing of the image menu.

3. Data can be retrieved by selecting option 1 to get a list of the available pictures. Once the list is displayed the prompt:

ENTER THE PICTURE NUMBER, OR ZERO FOR NO PICTURE ->

is displayed. Enter the number of the picture you want, and strike the return key to get the prompt:

TRANSMISSION TIME IS X.XX MINUTES

STRIKE THE F1 KEY AND ENTER FILENAME
WAITING...

4. At this point, you strike the **F1** key to get the prompt:

CAPTURE MODE ON. YOU WILL BE NOTIFIED WHEN RECEIPT IS COMPLETE.
PLEASE ENTER THE DISK FILENAME ->

Now enter the filename. Each file requires 128020 bytes of space on a diskette, so only two satellite data files can fit on one diskette. (SATDATA will display an error message and terminate the data retrieval process if insufficient space exists on the destination diskette.) If you decide that you do not want this picture, strike the **ESC** key to redisplay:

ENTER THE PICTURE NUMBER, OR ZERO FOR NO PICTURE ->

5. When all the data has been received, the main menu (as described in item 2 above) will again be displayed, with the top display line now indicating:

CAPTURE MODE OFF.

6. When finished retrieving data, select option **3** (Exit). AOS will automatically log you off, and you should see a NO CARRIER message. To exit SATDATA, strike the **F10** key, and get a DOS prompt (A>).

B. Displaying the Data

The data now on disk is ready to be displayed with a program called SATDSP. To start this program, type **satdsp<cr>**. SATDSP is the major program used to display and manipulate the satellite data. All the options in this program are menu driven with supplemental instructions given on the last several lines of every display screen. Appendix E contains figures of the display screens used in SATDSP with a brief description of their usage.

One of the advantages of using the Tecmar Graphics Master card is that it has the capabilities to drive a standard VCR monitor found in many field sites. Using a VCR monitor allows the image to be displayed with a gray scale instead of a color scale. Connection of the VCR monitor is done by patching the

RCA plug found on the back of the graphics card to a similar RCA plug (usually labeled video in) found on the VCR.

The default color scale used by SATDSP upon startup is stored in the disk file SATSCALE. If an office finds a favorite color scale, this new scale can become the default by saving it in the disk file SATSCALE. The scale used to properly display an image on a VCR differs dramatically from a scale used on a color monitor. If SATSCALE does not exist on the disk, SATDSP will default to a scale valid for a VCR monitor; therefore, by just renaming the SATSCALE file to some other filename, a scale valid for a VCR monitor automatically appears.

C. Automating Data Retrieval

SATDATA has been designed such that it can be used to automatically retrieve satellite data by providing command line arguments. The expanded format for starting the SATDATA program is:

A>satdata <filename> <picture number>

The filename is the disk file where the retrieved data will be stored. The picture number is the specific number which will be given the WRH AOS computer when the: ENTER THE PICTURE NUMBER, OR ZERO FOR NO PICTURE -> prompt is received. If both the filename and the picture number are input on the command, SATDATA will assume no operator is present and only display status messages on the screen as events are executed. If only the filename is given on the command line, a user is assumed to be present and will need to interact with the WRH AOS system as usual, except after the F1 key is entered, no filename needs to be given.

As an aid to automating data retrieval, a simple timer program has been developed to wait for a specified time. Once this specified time occurs, program control returns back to DOS so the SATDATA program can be executed. The timer program is called Wait and must receive a command line argument of the time to wait for; thus, the general format is:

A>wait hh:mm

Both SATDATA and WAIT can be combined into a batch file to automate data retrieval. The batch file would look something like:

```
echo off
wait 0:30
satdata file1 1
wait 1:30
satdata file2a 2
satdata file2b 1
wait 2:30
satdata file3 1
```

Upon completion, the disk would contain four satellite data files (file1, file2a, file2b and file3) which are ready for display.

IV. SOFTWARE DOCUMENTATION

The documentation contained in this chapter is divided into three sub chapters corresponding to the system modules depicted in Figure 1. The software source code has been included only in the sub chapters where the reader may find it informative, or useful. All of the source code is available in Western Region Headquarters, Scientific Services Division upon request.

The versions of software used in the development of these programs are:

IBM Personal Computer

MS-DOS	Version 3.10
MS-DOS Linker	Version 2.30
Lattice C Compiler	Version 3.00
IBM Macro Assembler	Version 1.00

Data General Eclipse S230

Advanced Operating System	Version 7.01
FORTRAN V AOS Compiler	Version 6.16
DG AOS Link	Version 7.00
DG AOS Macro Assembler	Version 7.00

A. The A/D Converter Software (SATCOMMS)

Functionally, this software module initializes the asynchronous communication and A/D converter hardware, and starts an infinite processing loop for collecting data. Inside this processing loop the start of a new picture is found, data are collected from the A/D converter hardware, and the data are compressed and transmitted to the dissemination computer. Figure 2 depicts a flowchart for this program which is called SATCOMMS. All assembly language routines used in SATCOMMS are found in section B, Remote User Software.

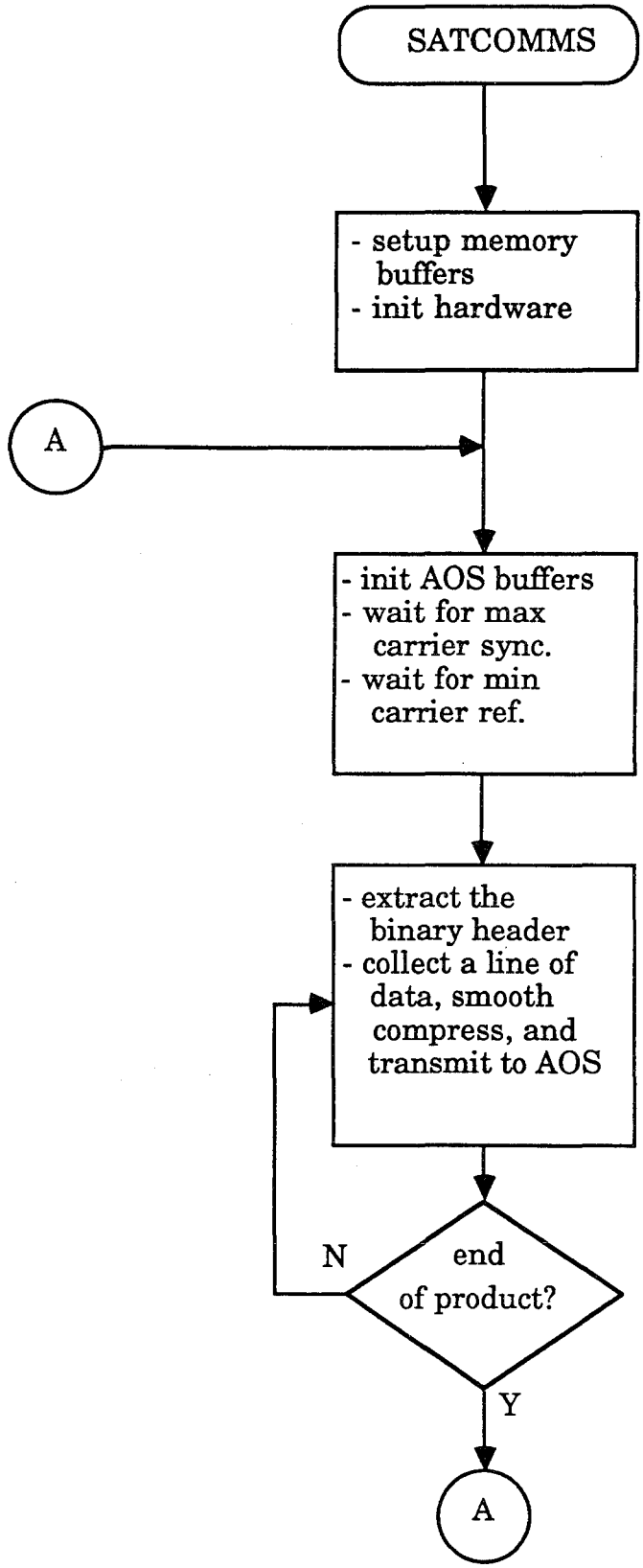


FIGURE 2 - SATCOMMS flowchart.

```
/*      date:  March 1987
        version:  1.0
        by:  Glen W. Sampson

Purpose:
  This program monitors the input buffer of the DT2814 A/D converter card,
filters the data down into 640x400x16 pixels of information, does a run length
data compression on the filtered data, and transmits the processed data out
to a dissemination computer.

Exits:
  This program monitors the keyboard for input of a CTRL C, which causes
termination.

Created by:
        (version 3.00 Lattice C compiler)
        C>lc -cu -md satcomms
        C>link cd satcomms func, satcomms, con, lcd
*/

/* parameters */
#define FILTER          1                       /* 1 - data is filtered, 0 - no filtering */
#define STORE           0                       /* 1 - write data to disk, 0 - no data written to disk */
#define ERROR           -1                      /* DOS error return code */
#define INSIZE          32765                   /* input buffer size */
#define INALLOC         32766                   /* since the compiler is too dense to figure insize + 1, we must
size */
#define OUTSIZE         65532                   /* output buffer size */
#define INBASE          760                     /* I/O base port adrs for A/D converter, 220H */
#define OUTBASE         1016                    /* alternate async adaptr base adrs, 2f8H */
#define NSIZ            30                      /* number of bytes required to flush AOS input buffer */

/* sampling parameters */
#define MAXSYNC         2300                    /* samples required for max sync ref detection */
#define MINSYNC         2334                    /* min sync samples required */
#define HI_MAX          255                     /* highest maximum value possible */
#define LO_MAX          175                     /* lowest maximum value */
#define HI_MIN          50                      /* highest minimum value */
#define LO_MIN          0                       /* lowest minimum value possible */
#define SAMP_LN         1500                    /* number of samples in a data line with the sync */
#define VERT_LN         640                     /* number of vertical lines on the display */
#define HORZ_LN         400                     /* number of horizontal lines on the display */

/* filtering parameters */
#define WGHT            10                      /* center pixel weight */
#define DIV             18                      /* divisor = wght + 8 */
#define ROUND           9                       /* roundup integer = div / 2 */
#define BLK_CUT         170                     /* annotation filter, black < BLK_CUT */
#include        "fcntl.h"                       /* get the standard dos file parameters */

/* global variables */
unsigned int optr;                              /* obuf[] output data indice */
```

```
unsigned int tx_ptr;                            /* obuf[] transmit indice */
int prodflg;                                    /* extrn variable for func.asm (used in voice synthesis */
int dates[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
char ibuf[INALLOC];                             /* digital data input buffer */
char *obuf;                                     /* satellite product output buffer */
char *iptr;                                     /* pointer to the input buffer */
char buf1[VERT_LN], buf2[VERT_LN], buf3[SAMP_LN], tbuf[SAMP_LN];       /* input buffers */
char hdr[SAMP_LN];

void main()
{
        extern int linenum;                             /* line counter defined in func.asm for use by ISR */
        unsigned int i, k;                              /* general purpose counters */
        int line;                                       /* line number currently being processed */
        int fd;                                         /* file descriptor */
        int end;                                        /* indice where actual data should end in tbuf[] */
        int skip;                                       /* sync samples to skip */
        char c1, c2;                                    /* temporary char variables */
        char inc();                                     /* character function declaration */
        char *getmem();                                 /* declare pointer function */
        char drver_ln[2];                               /* line number from the A/D device driver */
        char sync_ok;                                   /* sync ok flag , 0 - true 1- false */

/* establish our memory buffers */
        obuf = getmem ((unsigned) OUTSIZE + 3);         /* allocate output buffer */
        if (obuf == 0) {                                /* any errors? */
                printf("Memory allocation error\n");
                exit();                                 /* this error is fatal */
        }

/* initialize the Data Translation card and the async port */
        setcom (0,131);                                 /* initialize the async card */
        set_td (obuf);                                  /* setup the async int for quick enable */
        outp (INBASE, 0);                               /* start init by writing a zero to control reg. */
        delay (3);                                      /* delay 0.15 secs */
        c1 = inp (INBASE + 1);                          /* clear the data register */
        c2 = inp (INBASE + 1);

/* processing loop */
        for (;;) {
        /* initialize the systems */
                linenum = 0;                            /* reset line number to zero */
                tx_ptr = 0;                             /* init transmit pointer */
                iptr = &ibuf[0];                        /* set input pointer to the beginning of buffer */
                obuf[0] = 255;                          /* set init file size to 64K */
                obuf[1] = 255;
                i = OUTBASE + 5;                        /* line control reg */
                for (k = 0; k < 32; k++) {              /* ensure AOS is synced up */
                        while (!(inp(i) & 32));
                        outp(OUTBASE, 0);               /* send AOS some nulls */
                }
```

```
        /* sync up to the carrier signal */
                printf("max carrier ref search ");
                for (k = 0; k < 35; k++)                /* check max carrier ref, or receiver phasing */
                        wait (MAXSYNC, HI_MAX, LO_MAX);
                printf("/ min carrier ref search\n");
                wait (MINSYNC, HI_MIN, LO_MIN);         /* wait for the first ref */
                wait (MINSYNC, HI_MIN, LO_MIN);         /* wait for the second ref */
                satdgtl (&ibuf[0], &ibuf[INSIZE]);      /* enable IBM interrupts */
                outp (INBASE, 80);                      /* off we go, enable the DT2814 interrupts */

        /* extract the header information */
                for (line = 1; line < 3; line++) {
                        while (line > linenum);                         /* wait for a line of data */
                        for (i = 0; i < SAMP_LN; i++)                   /* grab the line of data */
                                tbuf[i] = inc();
                        inc();                                          /* skip the line number */
                        inc();
                        if (line == 1) {                                /* find the sync offset on the first line *
                                for (i = 0; tbuf[i] <= HI_MIN && i < SAMP_LN; i++);
                                skip = i + 25;                          /* calculate the end of max sync pulses */
                                printf("sync offset:  %d\n", skip);     /* display the sync offset */
                                for (i = 0; i < SAMP_LN; i++)
                                        buf3[i] = (~tbuf[i]) >> 1;              /* divide by 2 for a 2 line average
                        }
                        else {                                          /* must be the second line */
                                for (i = 0; i < SAMP_LN; i++) {         /* average the two lines */
                                        tbuf[i] = ((~tbuf[i]) >> 1) + buf3[i];
                                        hdr[i] = tbuf[i];
                                }
hdr[0] = skip;
hdr[1] = 0;
                        }
                }
                if (header (&tbuf[0], skip, SAMP_LN) > 3) {             /* determinate ASCII hdr and place in obuf[
                        reset();                                        /* error - disable int */
                        outp (INBASE, 0);                               /* disable DT2814 int */
                        printf ("Header error - tossing the current product\n");
                        delay (3);                                      /* pause */
                        inp (INBASE + 1);                               /* clear the DT2814 data regs */
                        inp (INBASE + 1);
                        continue;                                       /* wait for another product */
                }

        /* data collection, compression and transmission */
                for (; line < 392 && optr < OUTSIZE; line++) {
                        while (line > linenum);                 /* wait for a line of data */
/*                      printf("line = %x", line);               display line number */
                        sync_ok = 1;                            /* init sync flag to false */
                        for (i = 0; i < skip; i++, c1 = inc())  /* skip the sync */
                                if (sync_ok && c1 > LO_MAX)     /* is the max sync pulse present? */
                                        sync_ok = 0;            /* yes - set flag that sync is ok */
                        if (sync_ok) {                          /* is sync_ok still false */
```

```
                printf("lost sync, line = %d^G", line);
                break;                          /* exit and transmit nulls to finish product */
        }

/* filter data down */
        for (k = 0; i < SAMP_LN; i++) {
                c1 = inc();                     /* grab a byte of data */
                if (obuf[12] == 'E' && k < VERT_LN && line >= 25)
                        tbuf[k++] = ((~c1) & 240);      /* no line filter on ern sectors */
                else if (i % 2 && i % 19 && k < VERT_LN)/* grab 1 out of 2 bytes */
                        tbuf[k++] = ((~c1) & 240);      /* invert and isolate the first nibble */
        }
        end = k;                                /* save end of data */
        drver_ln[0] = inc();                    /* grab the line number */
        drver_ln[1] = inc();

/* smooth the data */
      #if FILTER                                        /* optional compile feature */
        if (line == 3 || line == 25)                    /* top of a boundary, save but don't filter
                movmem (&tbuf[0], &buf1[0], end);
        else if (line == 4 || line == 26) {             /* line below a boundary, save for filterin
                movmem (&tbuf[0], &buf2[0], end);
                continue;                               /* get the next line for filtering */
        }
        else if (line > 13 && line < 26)                /* an annotation line? */
                for (k = 0; k < end; k++)               /* yes - force black or white */
                        if (tbuf[k] > BLK_CUT)
                                tbuf[k] = 240;          /* force to white */
                        else
                                tbuf[k] = 0;            /* force to black */
        else {                                          /* regular line, 9 point filter */
                movmem (&tbuf[0], &buf3[0], end);       /* save line */
                filter (end);                           /* filter buf2[] data */
                if (line == 13) {                       /* boundary end? */
                        compress (end);                 /* save filtered data */
                        k = VERT_LN - end;
                        if (k)
                                obuf[optr++] = k;       /* fill remainder of line with nulls */
                        obuf[optr++] = 0;               /* terminate the line */
                        obuf[optr++] = 0;
                        movmem (&buf3[0], &tbuf[0], end);
                }
        }
      #endif

/* compress the data and finish the line */
        compress (end);                         /* compress data into obuf[] */
        k = VERT_LN - end;                      /* find the amount of space left */
        if (k)                                  /* fill remaining space with nulls */
                obuf[optr++] = k;
        obuf[optr++] = 0;                       /* terminate line with two nulls */
        obuf[optr++] = 0;
```

```
                /* ensure data is being transmitted out of the PC */
                i = OUTBASE + 1;                        /* get the int enable reg port */
                c1 = inp (i) & 2;                       /* check if the OUT2 bit is set */
                if (tx_ptr != optr && !c1)              /* does the int need to be reenabled? */
                        outp (i, 2);                    /* reenable the td int */

                /* check keyboard for possible abort */
                if ((c1 = getchr()) == 27)              /* check for an ESC key */
                        break;
        }

/* product finished, stop the interrupts */
        reset();                                /* reenable the clock, disable the A/D */
        outp (INBASE, 0);                       /* disable DT2814 int */
        delay (3);                              /* wait for card to react */
        c1 = inp (INBASE + 1);                  /* clear DT2814 data registers */
        c2 = inp (INBASE + 1);

/* ensure the remainder is transmitted */
        i = OUTBASE + 1;                                /* calculate the status adrs */
        c1 = inp(i) & 2;                                /* ensure ints are enabled */
        if (tx_ptr != optr && !c1)
                outp (i, 2);
        while (tx_ptr != optr);                         /* wait for obuf[] to empty */

        obuf[0] = optr & 255;                           /* place the product size in the header */
        obuf[1] = optr >> 8;
        printf("product size:  %x%02xH\n", obuf[1], obuf[0]);
        i = OUTBASE + 5;                                /* line control reg */
        for (k = 0; k < 2; k++) {                       /* transmit the product size as the last two bytes
                while (!(inp(i) & 32));                 /* wait for the finish */
                outp (OUTBASE, obuf[k]);
        }
        for (k = 0; k < NSIZ; k++) {                    /* flush the buffer in AOS */
                while (!(inp(i) & 32));                 /* wait for the last char */
                outp (OUTBASE, 0);                      /* transmit nulls */
        }

/* write the data out to disk */
     #if STORE
        if ((fd = dcreat ("satdata", 0)) == ERROR)      /* satelite product */
                printf("Unable to open file\n");
        if (dwrite(fd, &obuf[0], optr) == ERROR)
                printf("Unable to write data\n");
        dclose (fd);

        fd = dcreat ("header", 0);                      /* binary header */
        if (dwrite (fd, &hdr[0], SAMP_LN) == ERROR)
                printf("write error on header file\n");
        dclose (fd);
      #endif
```

```
                printf("\n");                                   /* put a blank line on the display */
        }
        exit();
}
/*****************************************************************************
*                                                                            *
*                              F U N C T I O N S                             *
*                                                                            *
*****************************************************************************/

/*----------------------------< start of inc >-------------------------------*/
/*****  name: inc()
This routine maintains the output from the input buffer.  The input buffer is a
circular buffer and must wrap around when at the end. Inc() returns a char, with
the first 8 bits representing the digital voltage value on a scale from 00H to
FFH (0 to +5 volts).
global variables used:
        char *iptr                      ;buffer pointer
        char buf[]                      ;input data buffer
routine callers:
        main()
*/
char inc()
{
        char c1;                                                /* return value */

        c1 = *iptr++;                                           /* grab a value from the buffer */
        if (iptr >= &ibuf[INSIZE])                              /* past the end? */
                iptr = &ibuf[0];                                /* reset to the beginning */
        return(c1);                                             /* return value to the caller */
}
/*----------------------------< end of inc>----------------------------------*/

/*----------------------------< start of delay >-----------------------------*/
/*****  name:  delay (ticks)
This routine delays for the specified number of ticks, where one tick equals
0.05495 seconds.  The maximum number of ticks which you can delay for is 65535,
or one hour.  The normal return value is zero.  The timer error has an maximum
error value of one tick.
global variables used:
        none
routine callers:
        main
*/
delay (ticks)
unsigned int ticks;
{
        timer (&ticks);                                         /* setup the timer interrupts */
        while (ticks);                                          /* wait for the timer to expire */
        return (0);                                             /* return to caller */
}
/*----------------------------< end of delay >-------------------------------*/
```

```
/*----------------------------< start of wait >------------------------------*/
/*****  name:  wait (count, hi_value, lo_value)
This routine waits for a repetition of sample data in the specified value
boundaries to determine minimum and maximum carrier references.
global variables used:
        none
routine callers
        main
*/
wait (count, hi_value, lo_value)
int count;                                          /* number of values to find */
int hi_value;                                       /* upper value boundary */
int lo_value;                                       /* lower value boundary */
{
        int i;                                      /* values to date counter */
        char c1, c2;                                /* general character variables */
        char status;                                /* DT2814 status register */

/* processing loop to find values */
        for (i = count; i > 0; ) {
                outp (INBASE, 0);                   /* trigger sample */
                while (((status = inp (INBASE)) & 128) == 0);/* wait for sample */
                c1 = inp (INBASE + 1);              /* grab data */
                c2 = inp (INBASE + 1);
                if (c1 >= lo_value && c1 <= hi_value)   /* inside boundaries? */
                        i--;                        /* yes - decrement count */
                else
                        i = count;                  /* no - reset counter */
        }
        return(0);                                  /* return to caller */
}
/*----------------------------< end of wait >--------------------------------*/

/*----------------------------< start of filter >----------------------------*/
/*****  name:  filter (size)
This routine filters satellite data using a nine point averaging scheme.  Data
must be contained in buf1[], buf2[] and tbuf[], with the data in buf2[] being
the line which is filtered.  The filtered data is returned in tbuf[].
routine callers:
        main
global variables used:
        buf1[]          buf2[]          buf3[]
        tbuf[]
*/
filter (size)
int size;                                           /* line size of data */
{
        int total;                                  /* summation of the pixels */
        int i;                                      /* general counter */
        char *pt[9];                                /* pixel pointers */
```

```
/* init pointers */
        pt[0] = &buf1[0];
        pt[1] = &buf1[1];
        pt[2] = &buf1[2];
        pt[3] = &buf2[0];
        pt[4] = &buf2[1];                               /* the pointer we are filtering */
        pt[5] = &buf2[2];
        pt[6] = &buf3[0];
        pt[7] = &buf3[1];
        pt[8] = &buf3[2];
        tbuf[0] = buf2[0];                              /* place the end points in buffer */
        tbuf[size] = buf2[size];
        size = size - 1;                                /* don't filter the end point */

/* processing loop */
        for (i = 1; i < size; i++) {
                total = *pt[0] + *pt[1] + *pt[2] + *pt[3] + *pt[5] + *pt[6] + *pt[7] + *pt[8];
                total += (*pt[4]) * WGHT;
                total = (total + ROUND) / DIV;
                tbuf[i] = total & 240;                  /* isolate the bits */
                pt[0]++;
                pt[1]++;
                pt[2]++;
                pt[3]++;
                pt[4]++;
                pt[5]++;
                pt[6]++;
                pt[7]++;
                pt[8]++;
        }
        size++;                                         /* restore the line size */
        movmem (&buf2[0], &buf1[0], size);              /* shift the buffers up one */
        movmem (&buf3[0], &buf2[0], size);              /* buf3[] is now available */
        return(0);                                      /* return to caller */
}
/*----------------------------< end of filter >-----------------------------*/

/*----------------------------< start of compress >-------------------------*/
/*****  name:  compress (pts)
This routine does a run length compression on the pixels in a line of data.
routine callers:
        main
global variables used:
        *obuf           optr            tbuf[]
*/
compress (pts)
int pts;
{
        int i;                                          /* general counter */
        char count;                                     /* run length counter */

        for (i = 0; i < pts && optr < OUTSIZE; ) {      /* process the line */
```

```
                obuf[optr] = tbuf[i++];                 /* get a byte of data to compare */
                for (count = 1; count < 15 && i < pts; i++)
                        if (obuf[optr] == tbuf[i])      /* is the next byte equal */
                                count++;                /* yes - increment counter */
                        else
                                break;                  /* no - terminate count */
                obuf[optr++] |= count;                  /* put the count in right nibble */
        }
        return(0);                                      /* return to caller */
}
/*----------------------------< end of compress >----------------------------*/

/*----------------------------< start of header >----------------------------*/
/*****  name:  header (&buf, start, end)
This routine converts the binary portion of the digital satellite header into an
ASCII format for placing in the first 20 bytes of the digital satellite product.
routine callers:
        main
global variables used:
        obuf[]          optr
*/
header (buf, start, end)
char buf[];                                             /* buffer containing the digital header */
int start;                                              /* where the data starts in the buffer */
int end;                                                /* where the data ends in the buffer */
{
        int i, j;                                       /* general counters */
        int total;                                      /* averaging variable */
        int x1, x2, x3;                                 /* 0.125 pixel values */
        int day, year, leap;                            /* date variables */
        int res, type;                                  /* resolution and image type info */
        char headr[250];                                /* bit array */

/* parse the data down into bits for analysis */
        for (i = start, j = 1; i < end; j++) {
                total = 0;                              /* initialize */

                /* each bit is formed from 5.625 pixels */
                switch (j % 4) {
                    case 1:
                        total = buf[i++] + buf[i++] + buf[i++] + buf[i++] + buf[i++];
                        x1 = buf[i++] / 8;                          /* calculate the .33 */
                        total = x1 * 5 + total;                     /* complete bit */
                        break;
                    case 2:
                        total = x1 * 3 + buf[i++] + buf[i++] + buf[i++] + buf[i++] +buf[i++];
                        x2 = buf[i++] / 4;
                        total += x2;                                /* value complete */
                        break;
                    case 3:
                        total = x2 + buf[i++] + buf[i++] + buf[i++] + buf[i++] + buf[i++];
                        x3 = buf[i++] / 8;
```

```
                        total = x3 * 3 + total;
                        break;
                    case 0:
                        total = x3 + buf[i++] + buf[i++] + buf[i++] + buf[i++] + buf[i++];
                        break;
                    default:
                        printf("ERROR: processing binary data\n");
                        break;
                }
                if (total > 660)                                    /* determine the bit value */
                        headr[j] = 1;
                else
                        headr[j] = 0;
        }

/* fill the output buffer with the time and date */
        optr = 2;                                           /* skip the size locations */
        for (i = 0, j = 1; i < 4; i++)
                obuf[optr++] = (headr[j++]<<3 | headr[j++]<<2 | headr[j++]<<1 |headr[j++]) + '0';

/* fill in the day */
        for (i = 0, day = 0; i < 3; i++) {                  /* extract the julian date */
                day = day * 10;
                day = (headr[j++]<<3 | headr[j++]<<2 | headr[j++]<<1 | headr[j++]) + day;
        }
        for (i = 0, year= 0; i < 2; i++) {                  /* extract the last two digits of the year */
                year = year * 10;
                year = (headr[j++]<<3 | headr[j++]<<2 | headr[j++]<<1 | headr[j++]) + year;
        }
        for (i = 0; day > 0 && i < 13;) {                   /* calculate the month and day */
                i++;                                        /* i = month */
                if (i == 2 && year%4 == 0)                  /* leap yr calcultn thru 2099A.D. */
                        leap = 1;
                else
                        leap = 0;
                day = day - dates[i] - leap;                /* subtract a month */
        }
        day = day + dates[i] + leap;                        /* adjust back to the month */
        if (day < 10) {                                     /* fill the output buffer */
                obuf[optr++] = '0';                         /* single digits */
                obuf[optr++] = day + '0';
        }
        else {
                stci_d (&obuf[optr], day, 4);               /* two digits */
                optr += 2;
        }

        /* fill the two letters for the month */
        switch (i) {
                case 1:                                     /* January */
                        obuf[optr++] = 'J';
                        obuf[optr++] = 'A';
```

```
            break;
        case 2:                                 /* February */
            obuf[optr++] = 'F';
            obuf[optr++] = 'E';
            break;
        case 3:                                 /* March */
            obuf[optr++] = 'M';
            obuf[optr++] = 'R';
            break;
        case 4:                                 /* April */
            obuf[optr++] = 'A';
            obuf[optr++] = 'P';
            break;
        case 5:                                 /* May */
            obuf[optr++] = 'M';
            obuf[optr++] = 'Y';
            break;
        case 6:                                 /* June */
            obuf[optr++] = 'J';
            obuf[optr++] = 'N';
            break;
        case 7:                                 /* July */
            obuf[optr++] = 'J';
            obuf[optr++] = 'L';
            break;
        case 8:                                 /* August */
            obuf[optr++] = 'A';
            obuf[optr++] = 'U';
            break;
        case 9:                                 /* September */
            obuf[optr++] = 'S';
            obuf[optr++] = 'E';
            break;
        case 10:                                /* October */
            obuf[optr++] = 'O';
            obuf[optr++] = 'C';
            break;
        case 11:                                /* November */
            obuf[optr++] = 'N';
            obuf[optr++] = 'O';
            break;
        case 12:                                /* December */
            obuf[optr++] = 'D';
            obuf[optr++] = 'E';
        default:
            break;
}

/* fill in two digits for the year */
stci_d (&obuf[optr], year, 4);                  /* put the year in obuf */
optr += 2;
```

B. Dissemination Software (SATELLITE) Process

Software documentation for the dissemination module is not reproduced here but is available from Western Region Headquarters, Scientific Services Division upon request.

C. Remote User Software

The remote user software consists of two programs and one library of assembler functions. One of the programs (SATEXPND) expands the compressed data for the user without a Hayes compatible modem. The other program (SATDSP) displays and manipulates the data for the user. The assembly language library was developed to decrease the display execution times, and to provide MS-DOS and other related system calls not found in the C compiler libraries.

1. Data Retrieval (SATDATA)

SATDATA provides a means to dial-up the dissemination computer, request a product, receive a product, expand the satellite data for display and log off the dissemination computer system. Figure 3 contains a flowchart of these tasks.

```
        /* extract resolution and image type */
        res = (headr[j++]<<2 | headr[j++]<<1 | headr[j++]);     /* resolution */
        type = (headr[j++]<<2 | headr[j++]<<1 | headr[j++]);    /* image type */

        /* extract the enhancement curve and sector information */
        for (i = 0; i < 5; i++) {
            obuf[optr]=headr[j++]<<6 | headr[j++]<<5 | headr[j++]<<4 | headr[j++]<<3 | headr[j++]<<2 | headr[j++]<<1 |

            if (obuf[optr] == ' ')
                obuf[optr++] = '_';                             /* change spaces to a slash */
            else
                optr++;                                         /* update pointer */
        }
        obuf[optr++] = 0;                                       /* null the excess byte */
        obuf[optr++] = 0;
        obuf[optr++] = 0;

/* display the satellite header on the screen */
        for (i = 2, j = 0; i < 17; i++)
                if ((obuf[i] < 48 && obuf[i]) || (obuf[i] > 57 && obuf[i] < 65) || (obuf[i] > 90 && obuf[i] != 95)
                        obuf[i] = '?';
                        j++;                                    /* keep count of the ? */
                }
        printf("picture title:  %c%c%c%c_%c%c%c%c",obuf[2],obuf[3],obuf[4],obuf[5],obuf[6],obuf[7],obuf[8],obuf[9]
        printf("%c%c_%c%c_%c%c%c%c\n",obuf[10],obuf[11],obuf[12],obuf[13],obuf[14],obuf[15],obuf[16],obuf[17]);
        return(j);                                              /* return to caller */
}
```

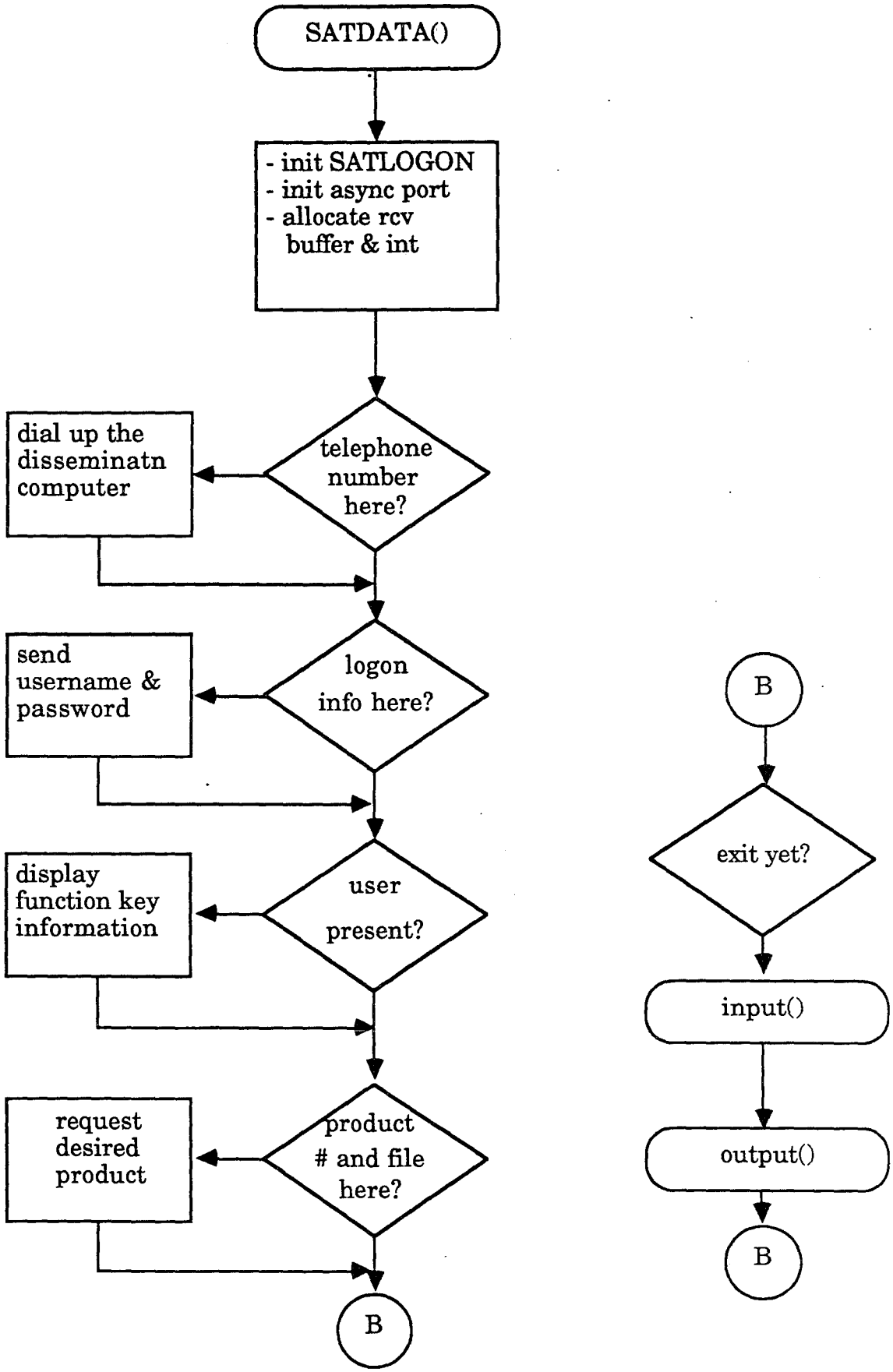


Figure 3 - SATDATA flowchart.

```
/*      date:  May 1987
        version:  1.0
        by:  Glen W. Sampson, NWS WRH SSD

Purpose:
  This program is designed to emulate a D211 terminal for an IBM to AOS
connection.

Exits:
  The F10 key will disable the async interrupts and return you to DOS.

Created by:
        (version 3.00 of Lattice C)
        C>lc -cu -md satdata
        C>link cd satdata func aos, satdata, con, lcd
*/

#include "stdio.h"
#include "fcntl.h"
#define INSIZE  (unsigned) 64000            /* input buffer size */
#define ERROR   -1                          /* DOS error return */
#define VERT_LN 640                         /* number of vertical lines in a display */
#define HORZ_LN 400                         /* number of horizontal lines */
#define TIMERUP 45                          /* telephone line check time out */

/* global variables */
struct logon {                              /* satlogon information */
        int comport;                        /* comms port; 0 - com1, 1 - com2 */
        char tele[40];                      /* telephone number */
        char usernm[40];                    /* AOS username */
        char passwrd[40];                   /* AOS password */
        char *out_file;                     /* output product filename pointer */
        char *prod_num;                     /* product number to request */
} x;
int prodflg;                                /* variables for func.asm */
int optr;
int tx_ptr;
int cflag;                                  /* continue on flag */
int ctrl;                                   /* ctrl bits */
int fd;                                     /* file descriptor */
int base;                                   /* base async port adrs */
char *receive;                              /* input buffer from AOS */
unsigned int out_ptr;                       /* pointers to receive */
extern unsigned int in_ptr;
extern unsigned int in_max, in_start;

void main(argc, argv)
int argc;                                   /* number of input arguments */
char **argv;                                /* argument pointers */
{
        int i;                                      /* general variable */
        int irq;                                    /* async int vector */
```

```
        char c;                                         /* general char variables */
        char s[40];                                     /* general string variable */
        char *getmem();                                 /* character pointers */

/* initialize the logon structure */
        if ((fd = dopen ("satlogon", O_RDONLY)) == ERROR) {     /* open the info disk file */
                x.comport = 0;                                  /* not there, set defaults - primary port */
                x.tele[0] = 0;                                  /* no telephone number */
                x.usernm[0] = 0;                                /* no username */
                x.passwrd[0] = 0;                               /* no password present */
        }
        else
                if (dread (fd, &x.comport, 122) == ERROR)       /* read the data into the structure */
                        fatal ("ERROR:  reading the satlogon file\n");
        dclose (fd);                                            /* ensure the file is closed */
        if (x.comport) {                                        /* init the base adrs */
                base = 760;                                     /* com2 port */
                irq = 3;
        }
        else {
                base = 1016;                                    /* com2 port */
                irq = 4;
        }
        if (argc > 1)                                           /* fill the output filename */
                x.out_file = argv[1];                           /* set pointer to the output filename */
        else
                x.out_file = 0;                                 /* null the pointer */
        if (argc > 2)                                           /* fill the product number requested */
                x.prod_num = argv[2];
        else
                x.prod_num = 0;                                 /* null the product number */

/* initialize the async. card and enable receive int */
/*      setcom (x.comport, 131);                                 set for 1200 baud, no parity, 1 stop, 8 bits */
        setcom (x.comport, 227);                               /* set for 9600 baud, no parity, 1 stop, 8 bits */
        if ((receive = getmem ((unsigned) INSIZE)) == 0) {
                putstr("memory allocation error^G\n");
                exit();
        }
        setaos (base, INSIZE, irq, &receive[0]);        /* enable interrupts */

/* automatic telephone dialing */
        if (x.tele[0]) {                                        /* dial if a telephone number exists */
            putstr("Is the telephone line ready for use (Y/N)? ");
            getstr (&s[0], 1);                                  /* get the users response */
            putchr('\r');                                       /* echo the users CR */
            putchr('\n');
            if (toupper (s[0]) == 'N')                          /* noooo - get out of this program then */
                fatal ("Execution terminated\n");
            else for (;;) {                                     /* loop until a carrier found */
                if (i = getchr()) {                             /* any keyboard activity */
                        i = i >> 8;                             /* isolate key code */
```

```
                if (i == 1)                     /* abort on the ESC key */
                    fatal ("Autodial aborted\n");
            }
            putstr("Executing the autodial\n");     /* inform the user */
            delay (19);                             /* enter the local command state */
            send ('+');
            send ('+');
            send ('+');
            delay (27);
            if (sendmodm ("AT V1 H0 E0 S7=25", "OK"))       /* setup our important parameters */
                continue;                           /* try again */
            s[0] = 0;                               /* flush string */
            strcat (&s[0], "AT DT");                /* setup first portion of dial command */
            strcat (&s[0], &x.tele[0]);             /* put in the telephone number */
            printf("Dialing %s\n", &x.tele[0]);     /* display dialing */
            if (sendmodm (&s[0], "CONNECT"))        /* dial the number */
                putstr("Unable to establish a carrier connect\n");
            else
                break;                              /* exit the autodial loop */
        }
    }

/* display instructions if not totally automated */
    if (x.prod_num == 0) {
            clear();                                /* clear the screen */
            putstr("Function keys are as follows:\n");
            putstr("  F1 - Capture data and store it on disk\n");
            putstr("  F10 - Return to DOS (exit AOS first)\n");
            putstr("  All other function keys are disabled\n");
            putstr("\nYou are currently on line to AOS\n");
    }

/* automatic logon */
    if (x.usernm[0] && x.passwrd[0]) {               /* ensure username and password present */
            putstr("Executing autologon\n");        /* inform user what is happening */
            delay (18);                             /* pause for the sloooo modem */
            send (138);                             /* send a character return */
            if (wait ("USERNAME:"))                 /* wait for the username colon */
                fatal ("Username prompt not found\n");
            sendstr (&x.usernm[0]);                 /* send the username to aos */
            if (wait ("PASSWORD:"))                 /* wait for the password colon */
                fatal ("Password prompt not found\n");
            sendstr (&x.passwrd[0]);                /* send the password */
    }

/* automate product retrieval */
    if (x.out_file && x.prod_num) {                 /* if info present, automatically retrieve the product */
            printf("Requesting product number %s\n", x.prod_num);
            if (wait(">"))                          /* wait for the prompt */
                fatal ("Main menu prompt not found\n");
            sendstr("1");                           /* request the product listing */
            if (wait (">"))                         /* wait for the prompt again */
```

```
                fatal ("Request prompt not found\n");
            sendstr(x.prod_num);                    /* send the product number */
            printf("Disk data file is %s\n", x.out_file);
            fd = dcreatx (x.out_file, 0);           /* open the requested file */
            if (fd == ERROR) {
                printf("  PREVIOUS FILE (%s) DESTROYED\n", x.out_file);
                if ((fd = dcreat (x.out_file, 0)) == ERROR)
                    fatal ("Output file create error\n");
            }
            delay (91);                             /* pause 5 secs for AOS to send Waiting... msg */
            while (inc(&c) == 0);                   /* clear out the input buffer */
            send (32);                              /* send a space to AOS to start transmission */
            ctrl = 1;                               /* set store flag */
            output();                               /* grab the product */
            if (wait(">"))                          /* wait for the prompt */
                fatal ("Main menu prompt not found\n");
            sendstr ("3");                          /* select the exit option */
            delay (182);                            /* pause 10 secs for the AOS logoff to come in */
            fatal ("Execution terminated\n");       /* return to DOS */
        }

/* processing loop, forever */
        ctrl = 0;                                   /* init control bits off */
        for(cflag = 1; cflag;) {
            input ();                               /* handle keyboard actions */
            output ();                              /* handle AOS responses */
        }
        fatal ("Execution terminated\n");           /* return to DOS */
}

/*****************************************************************************
*                                                                            *
*                            F U N C T I O N S                               *
*                                                                            *
*****************************************************************************/

/*---------------------------< start of input >-----------------------------*/
/*****  name:  input()
  This routine handles the IBM keyboard portion of the AOS interface
*/
input()
{
        char c1, c2;                                /* ASCII and key code info */
        char s1[40], s2[40];                        /* keyboard input string */
        int i;                                      /* general variable */

        if ( !(i = getchr()))                       /* check if there is a character */
            return(0);                              /* no input - return to caller */

/* process the input */
        c1 = i >> 8;                                /* key code (scan code) info */
        c2 = i & 255;                               /* ASCII info */
```

```
/* ASCII characters */
if (c1 < 59 || c1 > 68) {                       /* if not a function key */
        if (c2 >= 32 && c2 <= 126)
                send (c2);                              /* yes - send to AOS */

        /* control characters */
        else
                switch (c2) {
                    case 13:                    /* CR or LF */
                    case 10:
                        send (138);
                        break;
                    case 1:
                    case 2:
                    case 3:
                    case 4:
                    case 5:
                    case 6:
                    case 7:
                    case 9:
                    case 11:
                    case 12:
                    case 14:
                    case 15:
                    case 16:
                    case 17:
                    case 18:
                    case 19:
                    case 20:
                    case 21:
                    case 22:
                        c2 |= 128;              /* set the high bit */
                        send (c2);
                        break;
                    case 27:                    /* ESC key */
                        send (27);              /* send an ESC */
                        break;
                    case 8:
                        send (255);             /* send a OFFH */
                        break;
                    default:
                        break;
                }
}

/* function keys */
else
        switch (c1) {                                   /* user the scan code to determine action */
                case 68:                                /* function key */
                        cflag = 0;                      /* terminate forever loop */
                        break;
```

```
                        case 60:                                /* erase the screen */
                                clear();
                                out_ptr = in_ptr;               /* toss anything in the buffer */
                                send (138);                     /* send an LF */
                                break;                          /* return to caller */
                        case 59:
                                clear();
                                display (0, 0, 31, "Capture mode on.  You will be notified when receipt is complete
                                out_ptr = in_ptr;
                                if (x.out_file) {               /* filename already available? */
                                        printf("\nDisk data file is %s\n", x.out_file);
                                        if (disksp(x.out_file))
                                                fatal ("Disk space exhausted\n");
                                        fd = dcreatx (x.out_file);
                                        if (fd == ERROR) {
                                                printf("  PREVIOUS FILE (%s) DESTROYED\n", x.out_file);
                                                if ((fd = dcreat (x.out_file, 0)) == ERROR)
                                                        fatal ("Output file create error\n");
                                        }
                                }
                                else                                    /* user supplies filename */
                                        do {                            /* prompt the user for a filename */
                                                putstr("\nPlease enter the disk filename -> ");
                                                gets (&s1[0]);
                                                if (disksp(&s1[0]))     /* check disk space */
                                                        fatal ("Disk space exhausted\n");
                                                fd = dcreatx (&s1[0], 0);
                                                if (fd == ERROR) {      /* file already exists, overwrite? */
                                                        putstr("  FILE ALREADY EXISTS, OVERWRITE (Y/N)? ");
                                                        gets(&s2[0]);
                                                        if (toupper(s2[0]) == 'Y')
                                                                if ((fd = dcreat (&s1[0], 0)) == ERROR)
                                                                        fatal ("Output file create error^G\n");
                                                }
                                        } while (fd == ERROR);          /* loop until a valid fd is found */
                                ctrl = 1
;                       /* set control bits for capture */
                                send (32);                      /* send a space to start transmission */
                                break;
                        default:
                                putchr(7);                      /* beep for any other keys */
                                break;
                }
        return(0);
}

/*----------------------------< start of send >------------------------------*/
/*****  name:  send (c)
  This routine sends a character to AOS.
*/
send(c)
char c;                                 /* character to send */
```

```
{
        int x;                                  /* line status reg */

        x = base + 5;                           /* line status reg port */
        while ( !(inp (x) & 32));               /* wait for UART to clear */
        outp (base, c);                         /* send out the char */
        return(0);                              /* return to caller */
}
/*----------------------------< end of send >-------------------------------*/

/*----------------------------< start of output >---------------------------*/
/*****  name:  output ()
  This routine receives characters from AOS and displays them on the screen.
*/
output()
{
        int cnt;                                /* byte counter */
        int i, j;                               /* general counters */
        int line;                               /* line number */
        int val;                                /* value of the pixel */
        char cont;                              /* continue on flag */
        char c, c1, c2;                         /* buffer characters */
        char hdr[20];                           /* product header */
        char *getml();                          /* memory allocation pointer */
        char *buf_out, *buf_out1, *buf_out2;    /* output buffer pointers */
        char *obuf;                             /* output buffer pointers */

        /* check if data needs to be stored on disk */
        if (ctrl & 1) {                         /* in capture mode? */
            cont = 1;                           /* set flag to continue */
            for (i = 0; i < 20; i++)            /* grab the product header first */
                if (inc(&c) == 0)
                        hdr[i] = c;             /* put character in buffer */
                                                /* assume header is always here, so don't count nulls */
                else {
                        i--;                    /* no character adjust pointer */
                       continue;
                }
            if (dwrite (fd, &hdr[0], i) == ERROR)
                putstr("write error on output buffer header\n");

            /* process the product data */
            if ((buf_out = getml(128640L)) == 0)        /* allocate output memory */
                fatal ("ERROR:  allocating data output memory\n");
            buf_out1 = buf_out;                         /* setup buffer pointers */
            buf_out2 = buf_out + 64000L + VERT_LN/2;
            obuf = buf_out;
            line = 0;                           /* initialize line counter */
            while (cont && line < HORZ_LN) {    /* process the remaining data */
                line++;                         /* update the line counter */
                for (i = 0; i < VERT_LN;) {
                        while(inc(&c));                 /* wait for a character */
```

```
            if (c) {                        /* keep track of the nulls */
                    cnt = 0;                /* reset the counter */
                    j = c & 15;             /* isolate the count */
                    val = c >> 4;           /* isolate the pixel value */
                    for (; j > 0 && i < VERT_LN; j--, i++)
                            obuf[i] = val;  /* expand the data */
            }
            else {
                    cnt++;                  /* keep track of the null count */
                    if (cnt > 4) {
                            putchr(7);
                            putchr(7);
                            cont = 0;
                            break;
                    }
            }
    }
    if (cont != 1)                          /* ensure data is available */
            continue;                       /* no - exit loop */

    /* expand the data out */
    while (inc(&c1));                       /* skip two nulls */
    while (inc(&c2));
    if (c1 || c2) {                         /* if not null, an error has occurred */
            printf("Data error on line %d\n", line);
            c1 = c2;                        /* get ready to check the next byte */
            for (;;) {                      /* loop until two nulls are found */
                    inc (&c2);              /* grab the next byte of data */
                    if (!c1 && !c2)         /* two nulls yet? */
                            break;          /* yes - continue on */
                    c1 = c2;                /* no - get ready for next byte */
            }
    }
    cnt += 2;                               /* update the null counter */

    if (line % 2) {                         /* determine the proper output buffer */
            pack (buf_out1);                /* pack the data */
            buf_out1 += VERT_LN/2;
            obuf = buf_out2;                /* switch buffers */
    }
    else {
            pack (buf_out2);
            buf_out2 += VERT_LN/2;
            obuf = buf_out1;
    }
}

/* write the data out to disk */
obuf = buf_out + 64000L;                    /* use obuf as a temp pointer to buffer end */
for (; buf_out1 < obuf;)                    /* fill remainder with nulls */
    *buf_out1++ = 0;
obuf = buf_out + 128000L + VERT_LN/2;
```

```
        for (; buf_out2 < obuf;)
            *buf_out2++ = 0;
        if (dwrite (fd, buf_out, (unsigned) 64000) == ERROR)
            putstr("write error on the first output buffer\n");
        buf_out2 = buf_out + 64000L + VERT_LN/2;    /* reset buffer 2 pointer */
        if (dwrite (fd, buf_out2, (unsigned) 64000) == ERROR)
            putstr("write error on the second output buffer\n");
        dclose (fd);                        /* close the output file */

        rlsml (buf_out, 128640L);           /* release the output memory */
        ctrl = 0;                           /* reset the capture flag */
        if (x.prod_num == 0) {              /* if user present, clear screen and display info */
            clear();                            /* clear the screen */
            display (0, 0, 31, "Capture mode off                                                    ");
        }
        send (138);                         /* send a carriage return to AOS */
    }

    /* display AOS input */
    if (out_ptr != in_ptr) {
            inc (&c);                       /* grab a character from buffer */
            if (c >= 32 && c <= 126)
                    putchr(c);              /* display character */
            else
                    switch (c) {            /* control characters */
                        case 10:                    /* CR */
                            putchr('\r');
                            putchr('\n');
                            break;
                        case 7:                     /* bell */
                            putchr(7);
                            break;
                        case 25:                    /* backspace */
                            putchr(8);
                            break;
                        case 12:
                            clear();                /* clear the screen */
                            break;
                        default:
                            break;
                    }
    }
    return(0);
}
/*----------------------------< end of out >---------------------------------*/

/*----------------------------< start of inc >-------------------------------*/
/*****  name:  inc (&c)
  This routine maintains the *out_ptr.
*/
inc (c)
char c[];                                   /* adrs to place character */
```

```
{
        if (out_ptr != in_ptr) {
                c[0] = receive[out_ptr++];      /* grab a character */
                if (out_ptr > INSIZE)           /* check buffer pointer */
                        out_ptr = 0;            /* reset to buffer beginning */
                return(0);
        }
        return(1);                              /* return no char available */
}
/*----------------------------< end of inc >---------------------------------*/

/*----------------------------< start of fatal >-----------------------------*/
/*****  name:  fatal (&s)
  This routine displays an error message and returns to DOS.
*/
fatal (s)
char *s;                                        /* input message */
{
        char c;                                 /* general char variable */

        if (x.tele[0]) {                        /* if autodialed, hang up the phone */
                putstr("Hanging up the phone\n");
                delay (19);                     /* wait one second */
                send ('+');                     /* return the modem to the command state */
                send ('+');
                send ('+');
                delay (19);
                while (inc (&c) == 0);          /* clear out the buffer */
                sendmodm ("AT H", "OK");        /* hang up */
        }
        putstr(s);
        outp (base + 1, 0);                     /* disable async ints */
        exit(0);
        return(0);
}
/*----------------------------< end of fatal >-------------------------------*/

/*----------------------------< start of pack >------------------------------*/
/*****  name:  pack (&buffer)
  This routine takes two nibbles and stuffs them into one byte.
*/
pack (buffer)
char buffer[];
{
        int i, j;                               /* general counters */

        for (i = 0, j = 0; j < VERT_LN; i++, j++)
                buffer[i] = (buffer[j++] << 4) | buffer[j];
        return(0);                              /* all stuffed in */
}
/*----------------------------< end of pack >--------------------------------*/
```

```
/*--------------------------< start of wait >-------------------------------*/
/*****  name:  wait (c)
  This routine waits for a specified character to be received on an async card
before it returns to the caller.
*/
wait (c)
char *c;                                        /* chars to wait for */
{
        int count;                              /* timer counter */
        char aos_char;                          /* char received from aos */
        char *save;                             /* adrs of the start of the string */

        count = 546;                            /* wait 30 secs */
        timer (&count);                         /* set the timer going */
        save = c;                               /* init the save pointer */
        while (*c && count) {                   /* exit on string found, or timer expiration */
                while (inc(&aos_char) && count);/* get a character from the buffer */
                if (aos_char == *c)             /* a match? */
                        c++;                    /* yes increment pointer */
                else
                        c = save;               /* reset string pointer to beginning */
        }
        if (count == 0)
                return(1);                      /* return error on timer expiration */
        if (count > 2)
                count = 1;
        while (count);                          /* ensure timer has expired before returning */
        return (0);
}
/*--------------------------< end of wait >---------------------------------*/

/*--------------------------< start of sendstr >----------------------------*/
/*****  name: sendstr (s)
  This routine sends a string of characters followed by a carriage return out the
async port.
*/
sendstr (s)
char *s;                                        /* adrs of string to send */
{
        while (*s)                              /* send until a null is encountered */
                send (*s++);
        send (138);                             /* terminate with a carriage return */
        return(0);                              /* return to caller */
}
/*--------------------------< end of sendstr >------------------------------*/

/*--------------------------< start of delay >------------------------------*/
/*****  name:  delay (ticks)
  This routine delays for the specified number of ticks.
*/
delay (ticks)
int ticks;                                      /* number of ticks to delay, 18.2 ticks = 1 sec */
```

```
{
        timer (&ticks);                         /* set the timer going */
        while (ticks);                          /* wait for it to expire */
        return(0);
}
/*---------------------------< end of delay >-------------------------------*/

/*---------------------------< start of sendmodm >--------------------------*/
/*****  name:  sendmodm (str, response)
  This routine sends a string to the modem, and checks for a valid response.
*/
sendmodm (str, response)
char *str;                                      /* modem command string */
char *response;                                 /* response string to compare */
{
        int x;                                  /* return value */

        /* send out the command */
        while (*str)                            /* transmit the entire string */
                send (*str++);
        send (13);                              /* terminate with a carriage return */

        /* ensure the modem responds */
        x = wait (response);                    /* wait for the modem response */
        return (x);                             /* return the response code */
                                                /* 0 - ok, 1 - error */
}
/*---------------------------< end of sendmodm >----------------------------*/

/*---------------------------< start of disksp >----------------------------*/
/*****  name:  disksp()
  This routine ensures enough space is left on disk to save a satellite picture.
*/
disksp(str)
char str[];                                     /* filename string */
{
        int drive;                              /* drive number */
        struct diskinfo {
                unsigned short free;            /* number of free clusters */
                unsigned short cpd;             /* clusters per drive */
                unsigned short spc;             /* sectors per cluster */
                unsigned short bps;             /* bytes per sector */
        } info;
        long disk_sz;                           /* free space on the disk */
        int test;

        /* determine the drive number */
        if (str[1] == ':')                              /* drive preceeds the filename? */
                drive = toupper(str[0]) - 'A' + 1;      /* yes - extract info */
        else
                drive = 0;                              /* no - use the current drive */
        /* get the size */
```

```
        if ((test = getdfs (drive, &info)) == 0)
                disk_sz = (long) info.free * info.spc * info.bps;
        else
                disk_sz = 0;
        if (disk_sz < 128020L)
                return(1);
        return(0);
}
/*---------------------------< end of disksp >-----------------------------*/

/*---------------------------< start of getstr >---------------------------*/
/*****  name:  getstr (s, n)
This routine assembles a string of user input from the keyboard.  A return of
zero indicates input is ok, a return of one means the ESC key has been struck.
global variables used:
        none
routine callers:
        main()          record()        category()      playbk()
*/
getstr (s, n)
char s[];                               /* string to put input into */
int n;                                  /* number of characters user can input */
{
        int i, j;                               /* general counters */
        int temp1;                              /* temporary variable */
        int retval;                             /* function return value */
        char c1, c2;                            /* char variables */

        /* initialize string */
        for (i = 0; i < n; i++)
                s[i] = ' ';                             /* fill with spaces */

        retval = 0;                                     /* init return value */
        i = 0;                                          /* init indice */
        while (i <= n && retval == 0) {                 /* keyboard editing loop */
                j = 0;                                  /* init timer counter */
                while ( !(temp1 = getchr())) {          /* wait for a keyboard entry */
                        delay (2);                      /* delay 1/9 */
                        j++;                            /* inc counter */
                        if (j == (TIMERUP * 9)) {       /* time up yet */
                                temp1 = 283;            /* ESC key code */
                                break;
                        }
                }
                c1 = temp1 & 255;                       /* isolate ASCII value */
                c2 = (temp1 & 65280) >> 8;              /* isolate key code */
                if (c2 == 28)                           /* carriage return entered? */
                        break;                          /* exit editing loop */
                else
                        switch (c2) {                   /* process the key codes */
                            case 1:                     /* ESC key */
                                for (; i > 0; i--)      /* clear any entry */
```

```
                                putstr("\b \b");
                            retval = 1;             /* return to caller */
                            break;
                        case 14:                    /* backspace key */
                        case 75:                    /* left arrow key */
                            if (i > 0) {            /* can't go past starting point */
                                i--;                /* adjust pointer */
                                putchr(8);          /* move cursor */
                            }
                            break;
                        case 77:                    /* right arrow */
                            if (i < n) {            /* can't go past end */
                                putchr(s[i]);       /* move cursor forward */
                                i++;                /* adjust pointer */
                            }
                            break;
                        default:                    /* everything else */
                            if (c1 >= 32 && c1 <= 126 && i < n) {
                                c1 = tolower(c1);   /* convert to lower case */
                                putchr(c1);         /* echo input */
                                s[i] = c1;          /* save character */
                                i++;                /* update pointer */
                            }
                            break;
                    }
        }

        /* fill remaining string with spaces and clear user entry area */
        if (!retval)
                for (; i < n; s[i++] = ' ', putchr(' '));
        s[i] = 0;                                   /* terminate string */
        return(retval);                             /* return to caller */
}
/*---------------------------< end of getstr >------------------------------*/

/*---------------------------< start of putstr >----------------------------*/
/*****  name:  putstr(&s)
  This routine displays a string of characters onto the display.
*/
putstr (s)
char *s;                                            /* input string */
{
        while (*s) {
                if (*s == '\n')                     /* line feed? */
                        putchr('\r');               /* preceed with a carriage return */
                putchr(*s++);                       /* display until a null */
        }
        return(0);
}
/*---------------------------< end of putstr >------------------------------*/
```

```
/*      date:  April 1987
        version:  1.00
        by:  Glen W. Sampson

Purpose:
  This program is designed to create the satellite information file to be used
for automatic dial and logon to a Data General AOS system.  This information
file contains:  com1 or com2 designation, telephone number to dial, the AOS
username, and the AOS password.

Exits:
  The program exits after answering the last question.

Created by:
        (version 3.00 of Lattice C)
        C>lc -cu -md -v satinfo
        C>link cd satinfo func, satinfo, con, lcd
*/
#define ERROR   -1                              /* DOS error return code */

/* global variables (needed for func.asm) */
int prodflg;
int optr;
int tx_ptr;

void main()
{
        struct info {                           /* infomation structure */
                int comport;
                char tele[40];
                char usernm[40];
                char passwrd[40];
        } x;
        char s[40];                             /* general input string */
        int fd;                                 /* file descriptor */

/* print user instructions */
        clear();                                /* clear the display */
        printf("Please answer the questions as they appear.\n");
        printf("  ALL ENTRIES MUST BE LESS THAN 40 CHARACTERS\n\n\n");

/* get the user responses */
        printf("Is your Hayes modem setup as COM1 (Y/N)? ");
        gets (&s[0]);                           /* get the user response */
        if (toupper (s[0]) == 'Y')
                x.comport = 0;
        else {                                  /* default to com2, let the user know */
                x.comport = 1;
                printf("  Defaulting to COM2\n");
        }

        printf("Enter the telephone number to dial\n");
```

```
        printf("Include commas to pause (e.g. 8,8013289301)\n");
        printf("-> ");
        gets (&x.tele[0]);

        printf("Enter your username -> ");
        gets(&x.usernm[0]);

        printf("Enter your password -> ");
        gets(&x.passwrd[0]);

/* write the info out to disk */
        if ((fd = dcreat ("satlogon", 0)) == ERROR)
                fatal ("Create error on SATLOGON\n");
        if (dwrite (fd, &x, 122) == ERROR)
                fatal ("Write error on SATLOGON\n");
        printf("Information is now in the disk file SATLOGON\n");
        exit();
}
/*****************************************************************************
*                                                                            *
*                              F U N C T I O N S                             *
*                                                                            *
*****************************************************************************/

/*****  name:  fatal (&s)
  This routine displays an error message before exiting to DOS.
*/
fatal (s)
char *s;                                        /* input string adrs */
{
        printf("%s^G", s);
        exit();
        return(0);
}
```

2. Data Display (SATDSP)

Display of the satellite data is accomplished with SATDSP. SATDSP is completely menu driven and contains several lines of instructions at the bottom of every display screen to guide the user. Four main options are available for the user: 1. single picture display, 2. picture looping, 3. changing the color scale, and 4. exit the program. Each option is entirely contained in an individual function, so additional options can simply be added by adding another software function. The functions available in this program file are:

main	getstr	delay	mode
pixload	lowdsp	highdsp	enhance
erase	menu	onepix	loopix
colors	arrow	unarrow	dspscale
cursor			

Figure 4 contains the general flowchart for SATDSP.

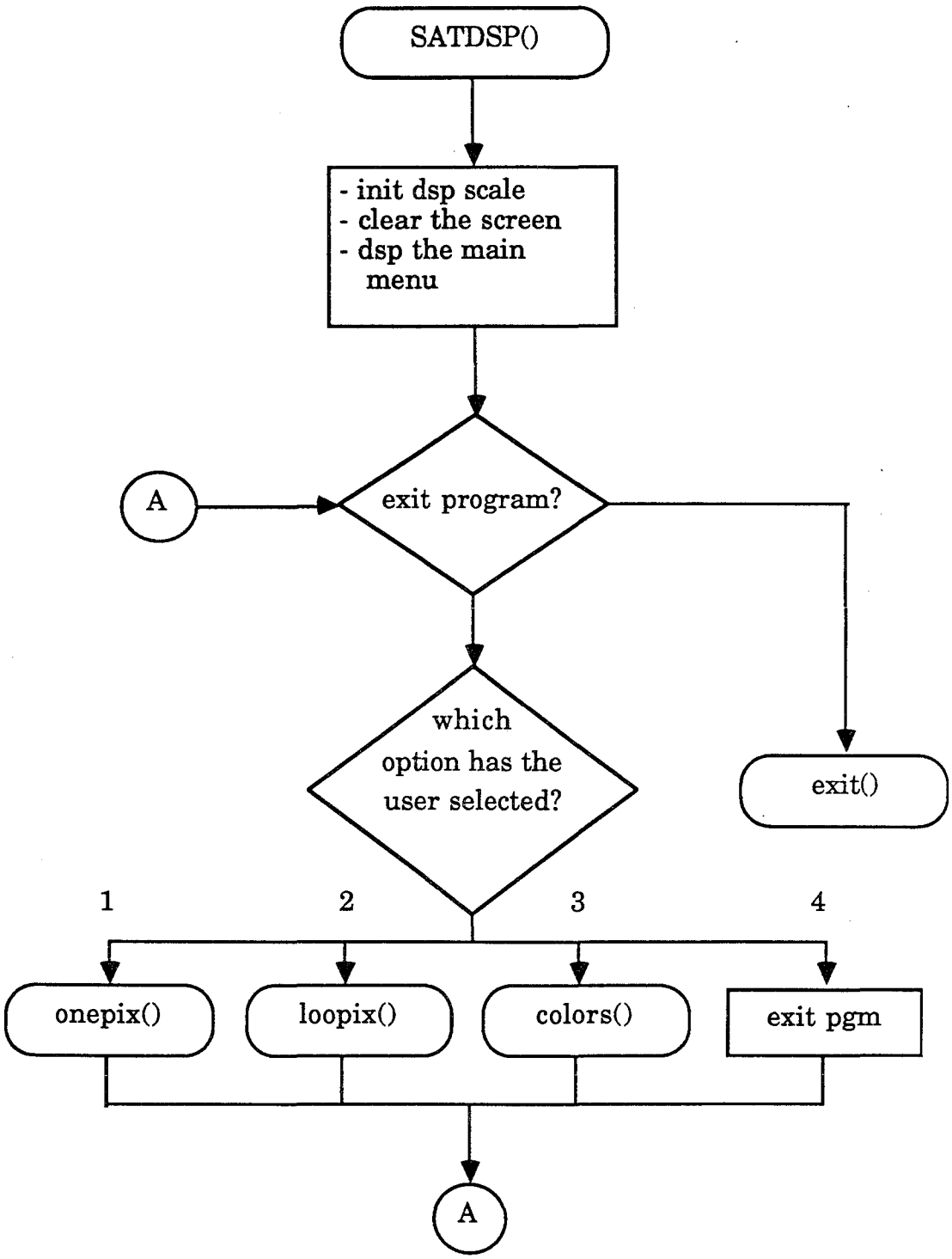


FIGURE 4 - SATDSP flowchart.

```
/*      date:  May 1987
        version:  1.0
        by:  Glen W. Sampson, NWS WRH SSD

Purpose:
  This program provides a method to display satellite data on a standard IBM color
display monitor using a Tecmar Graphic Master controller card.

Exits:
  The normal exit from this program is thru the ESC key.  Strike ESC to exit at
any time.

Created by:
        (version 3.00 of the Lattice C Compiler)
        C>lc -cu -md satdsp
        C>link cd satdsp func, satdsp, con, lcd
*/

/* get the include files loaded in */
#define ERROR   -1                              /* DOS error return code */
#define lorsltn 200                             /* number of horizontal lines in low resolution */
#define hirsltn 400                             /* number of lines in high resolution */
#include "fcntl.h"                              /* DOS file parameters */

/** EXTERNAL VARIABLES **/
/* graphic display modes */
int pixloop1[] = {0,0,0,32,1,2,112,100,6,127,15,184,160,227,0,31,24};   /* pix looping mode (sections 0 & 1) */
int pixloop2[] = {0,0,0,32,1,2,112,100,6,127,15,184,160,227,0,90,24};   /* sections 2 and 3 */
int pixdsp[] = {0,0,0,32,3,3,56,50,1,64,15,184,160,227,0,31,24};        /* high resolution interlace */
int alpha[] = {0,0,7,6,7,2,28,25,5,31,15,86,80,113,0,16,1};             /* alphanumeric mode */

/* display screen data format */
struct page {
        int row;                                /* row to display data on */
        int col;                                /* column to start data on */
        int attrib;                             /* attribute (color) of data line */
        char *line;                             /* pointer to the data line */
};

/* the main display menu */
struct page pag0[] = {
        0,28,2,"Department of Commerce",
        1,17,2,"National Oceanic and Atmospheric Administration",
        2,27,2,"National Weather Service",
        3,26,2,"Western Region Headquarters",
        6,25,15,"SATELLITE DATA DISPLAY SYSTEM",
        21,0,14,"Instructions:",
        21,15,6,"Enter the selection number.  Striking the ESC key at any time",
        22,0,6,"will abort the selection you are currently in, and return you to this menu.",
        9,20,10,"Main menu selections:",
        10,20,10,"1.  single satellite picture display",
        11,20,10,"2.  looped satellite picture displays",
```

```
        12,20,10,"3.  change the color curve",
        13,20,10,"4.  return to DOS",
        16,20,10,"Enter selection:  ",
        -1,-1,-1,""
};

/* single picture display menu */
struct page pag1[] = {
        0,1,15,"SINGLE PICTURE DISPLAY",
        21,0,14,"Instructions:",
        21,15,6,"Please answer the questions as prompted.  Strike the ESC key to",
        22,0,6,"stop the display and return to the main menu.  High resolution is 640x400x16",
        23,0,6,"with a screen flicker; low resolution is 640x200x16 with no screen flicker.",
        5,12,2,"0 - low resolution",
        6,12,2,"1 - high resolution",
        -1,-1,-1,""
};

/* looping a series of satellite pictures */
struct page pag2[] = {
        0,1,15,"LOOPED PICTURES DISPLAY",
        21,0,14,"Instructions:",
        21,15,6,"Please enter the satellite data filenames.  Strike return with no",
        22,0,6,"entry to terminate input.  Once the loop begins, use the ^X^Y keys to change",
        23,0,6,"the speed and the space bar to pause and restart.  Strike the ESC key to exit.",
        -1,-1,-1,""
};

/* changing the color scale display */
struct page pag3[] = {
        0,1,15,"CHANGE THE COLOR CURVE",
        21,0,14,"Instructions:",
        21,15,6,"Please answer the questions as prompted.  Strike the ESC key to",
        22,0,6,"abort and return to the main menu.",
        2,3,10,"Do you wish to load a previous",
        3,3,10,"color curve (Y/N)?  ",
        0,0,0,"",
        1,41,10,"Current Scale",
        1,66,10,"New Scale",
        3,42,2,"level  0",
        3,55,4,"warmest",
        3,64,2,"level  0",
        4,42,2,"level  1",
        4,64,2,"level  1",
        5,42,2,"level  2",
        5,64,2,"level  2",
        6,42,2,"level  3",
        6,64,2,"level  3",
        7,42,2,"level  4",
        7,64,2,"level  4",
        8,42,2,"level  5",
        8,64,2,"level  5",
```

```
        9,42,2,"level  6",
        9,64,2,"level  6",
        10,42,2,"level  7",
        10,64,2,"level  7",
        11,42,2,"level  8",
        11,64,2,"level  8",
        12,42,2,"level  9",
        12,64,2,"level  9",
        13,42,2,"level 10",
        13,64,2,"level 10",
        14,42,2,"level 11",
        14,64,2,"level 11",
        15,42,2,"level 12",
        15,64,2,"level 12",
        16,42,2,"level 13",
        16,64,2,"level 13",
        17,42,2,"level 14",
        17,64,2,"level 14",
        18,42,2,"level 15",
        18,55,3,"coolest",
        18,64,2,"level 15",
        19,66,2,"update",
        21,57,6,"Use the ^[* keys to",
        22,0,6,"parse through the colors, and the ^X^Y keys to change the level edited.  Strike",
        23,0,6,"the space bar to select a specific color.  Select 'update' when finished.",
        -1,-1,-1,""
};

/* other variables */
char scale[256];                        /* color scale array */
int prodflg;                            /* globals variables for func.asm */
int tx_ptr;
int optr;

main()
{
        int i, j;                       /* general purpose counters */
        int cflag;                      /* continue on flag */
        int fd;                         /* file descriptor */
        int temp1, temp2;               /* temporary variables */
        char s[15];                     /* general purpose string */

        /* initialize the color scale */
        fd = Dopen ("satscale", O_RDONLY);      /* open default file */
        if (Dread (fd, &scale[0], 256) == ERROR) {/* read in data if there */
                for (i = 0; i < 16; i++)        /* if error, default to standard scale */
                        scale[i] = i;
                for (i = 1; i < 16; i++) {      /* expand the scale to a 256 byte array */
                        temp1 = i << 4;
                        temp2 = scale[i] << 4;
                        for (j = 0; j < 16; j++)
                                scale[temp1 + j] = temp2 | scale[j];
```

```
            }
        }
        Dclose (fd);                            /* close the file */

        /* clear screen and display menu */
        mode (&alpha[0]);                       /* temporary alpha mode set */
        menu();

        /* processing loop */
        for (cflag = 1; cflag;) {
                getstr (&s[0], 1);              /* let user select */
                switch (s[0]) {                 /* determine the users request */
                    case '1':                   /* one picture display */
                        onepix();
                        menu();
                        break;
                    case '2':                   /* looped picture displays */
                        loopix();
                        menu();
                        break;
                    case '3':                   /* change the color curve */
                        colors();
                        menu();
                        break;
                    case '4':                   /* return to DOS */
                        cflag = 0;              /* change the continue on flag to exit */
                        break;
                    default:
                        printf("^G");           /* let the user hear any mistakes */
                        break;
                }
        }
        clear();
        display (0,0,11,"bye");                 /* notify user we are gone */
        exit();                                 /* normal exit */
        return(0);
}
/*****************************************************************************
*                                                                            *
*                            F U N C T I O N S                               *
*                                                                            *
*****************************************************************************/

/*****  name:  getstr (s, n)
This routine assembles a string of user input from the keyboard.  A return of
zero indicates input is ok, a return of one means the ESC key has been struck.
global variables used:
        none
routine callers:
        main()          onepix()        loopix()
*/
getstr (s, n)
```

```
char s[];                                   /* string to put input into */
int n;                                      /* number of chars user can input */
{
        int i;                                          /* general counter */
        int temp;                                       /* temporary variable */
        int retval;                                     /* function return value */
        char c1, c2;                                    /* char variables */

        /* initialize the string */
        for (i = 0; i < n; i++)
                s[i] = ' ';                             /* fill with spaces */

        retval = 0;                                     /* init the return val */
        i = 0;                                          /* init string indice */
        while (i <= n && retval == 0) {                 /* keyboard editing loop */
                while ( !(temp = getchr()));            /* wait for input */
                c1 = temp &255;                         /* isolate ASCII code */
                c2 = (temp & 65280) >> 8;               /* isolate the key code */
                if (c2 == 28)                           /* carriage return entered? */
                        break;                          /* exit input loop */
                else
                        switch (c2) {                   /* process the key codes */
                            case 1:                     /* ESC key */
                                for (; i > 0; i--)      /* clear any entry */
                                    printf("\b \b");
                                retval = 1;             /* let the caller know */
                                break;
                            case 14:                    /* backspace key */
                            case 75:                    /* left arrow key */
                                if (i > 0) {            /* can't go past starting point */
                                    i--;
                                    printf("\b");
                                }
                                break;
                            case 77:                    /* right arrow key */
                                if (i < n) {            /* can't go past the end */
                                    printf("%c", s[i]);
                                    i++;
                                }
                                break;
                            default:                    /* everything else */
                                if (c1 >= 32 && c1 <= 126 && i < n) {
                                    c1 = tolower(c1);   /* convert to lower case */
                                    printf("%c", c1);   /* echo input */
                                    s[i] = c1;          /* save the character */
                                    i++;                /* update pointer */
                                }
                                break;
                        }
        }

        /* fill remaining string with spaces and clear user entry area */
```

```
        if (!retval)
                for (; i < n && i > 0; s[i++] = ' ', printf(" "));
        s[i] = 0;                                       /* terminate the string */
        return (retval);                                /* return to caller */
}

/*****  name:  delay (ticks)
This routine delays for the specified number of ticks, where one tick equals
0.05495 seconds.  The maximum number of ticks which you can delay for is 65536,
or one hour.  The normal return value is zero.  The timer error has an maximum
error value of one tick.
global variables used:
        none
routine callers:
        loopix()
*/
delay (ticks)
unsigned int ticks;                             /* ticks to delay */
{
        int retval;                             /* return value */
        int temp;                               /* temporary variable */
        char c;                                 /* character variable */

        timer (&ticks);                                 /* setup the timer interrupts */
        retval = 0;
        while (ticks)                                   /* wait for timer to expire */
                if (temp = getchr())                    /* check for the esc key */
                        if ((c = temp & 255) == 27) {
                                retval = 1;             /* let the caller know */
                                ticks = 1;              /* cancel the timer */
                                break;                  /* exit timer loop */
                        }
        return (retval);                                /* return to caller */
}

/*****  name:  MODE(PARAM)
  This routine establishes the graphics mode for the color display.  All data
should be present in the graphics buffer before this function is called.
*/
mode(param)
int param[];
{
        int i, j;                               /* general counters */

        for (i = 13, j = 0; i >= 0; i--, j++) { /* plug in the registers */
                outp (916, i);                  /* set pointer reg - I/O port 394H */
                outp (917, param[j]);           /* put data in reg - I/O port 395H */
        }
        outp (921, param[j++]);                 /* set color select reg */
        outp (922, param[j++]);                 /* set the extended mode select reg */
        outp (920, param[j]);                   /* set the mode select reg - enable the display */
        delay (1);                              /* wait one tick, so we can display immediately upon return */
```

```
        return(0);                              /* normal return */
}

/*****  name:  pixload (fd, adrs, size, lines)
This routine reads a satellite data file into memory.
global variables used:
        none
routine callers:
        onepix()        loopix()
*/
pixload (fd, adrs, size, lines)
int fd;                                 /* satellite data file descriptor */
char **adrs;                            /* pointer where memory data will be buffered */
long *size;                             /* location of buffer size */
int lines;                              /* number of lines in display, 200 or 400 */
{
        int i;                                  /* general counters */
        long bufsz;                             /* size of the data buffer */
        char hdr[20];                           /* file header buffer */
        char *lsbrk();                          /* declare the pointer functions */
        char *buf, *buf_sv;                     /* allocated memory buffer pointer */

        /* read in the file header */
        if (Dread (fd, &hdr[0], 20) == ERROR)
                return(1);                              /* return one on errors */

        /* display the file header on the screen */
        display (19,24,7,"");                           /* move the cursor */
        printf("File header: %c%c%c%c_%c%c%c%c%c%c_",hdr[2],hdr[3],hdr[4],hdr[5],hdr[6],hdr[7],hdr[8],hdr[9],hdr[10
        printf("%c%c_%c%c%c",hdr[12],hdr[13],hdr[14],hdr[15],hdr[16]);

        /* allocate memory */
        bufsz = lines * 320L;                           /* calculate the buffer size */
        buf = lsbrk(bufsz);                             /* allocate the memory */
        if (buf == 0) {                                 /* memory available? */
                *size = 0;                              /* no - terminate data buffering */
                return(0);                              /* return to user normally */
        }
        else
                *size = bufsz;                          /* let caller know the size of the buffer */
        buf_sv = buf;                                   /* save the buffer starting adrs */

        /* read in the actual data */
        for (i = 0; i < lines; i += 200) {
                if (Dread (fd, buf, (unsigned) 64000) == ERROR) {
                        rbrk();                         /* release the memory */
                        return(1);
                }
                enhance (buf);                          /* enhance the picture */
                buf += 64000L;                          /* get the next buffer offset */
        }
```

```
        *adrs = buf_sv;                                 /* let the caller know where data is located */
        return(0);                                      /* return to caller */
}

/*****  name:  lowdsp (adrs)
This routine displays a low resolution (640x200x16) image.  The number of lines
input controls the resolution.  This routine assumes the proper amount of data
exists in the input buffer for display.
global variables used:
        none
routine callers:
        onepix()        loopix()
*/
lowdsp (adrs)
char *adrs;                             /* data buffer adrs */
{
        int i;                                          /* general counter */
        unsigned int offset;                            /* memory card offset */

        /* processing loop */
        offset = 0;                                                     /* initialize the pointer offset to beginni
        for (i = 0; i < lorsltn; i++, adrs += 320)
                if (i % 2) {
                        dspln (offset, (unsigned) 43008, adrs);         /* second section */
                        offset += 320;                                  /* update the offset */
                }
                else
                        dspln (offset, (unsigned) 40960, adrs);         /* first section */
        return(0);                                                      /* normal return */
}

/*****  name:  highdsp (adrs1)
This routine displays a high resolution graphic image (640x400x16).  The proper
amount of data is assumed to reside in the buffer upon entry.
global variables used:
        none
routine callers:
        onepix()
*/
highdsp (adrs1)
char *adrs1;                                    /* buffer pointer to data */
{
        int xmsr_on, xmsr_off;                                  /* xmsr register value with bit 6 on and off */
        char *adrs2;                                            /* second screen buffer adrs */
        int offset;                                             /* offset for graphic memory */
        int i;                                                  /* general counter */

        xmsr_on = pixdsp[15] | 64;                              /* set bit 6 on */
        xmsr_off = pixdsp[15] & 191;                            /* ensure bit 6 is off */
        adrs2 = adrs1 + 64000L;                                 /* calculate the start of the second buffer adrs */

        /* display the data */
```

```
        for (i = 0, offset = 0; i < hirsltn; i++)
                if (i % 2) {
                        if ((i - 1) % 4) {                      /* fill section 3 */
                                dspln (offset, (unsigned) 43008, adrs2);
                                offset += 320;                  /* set offset for the next group of lines */
                        }
                        else                                    /* fill section 1 */
                                dspln (offset, (unsigned) 43008, adrs2);
                        adrs2 += 320L;                          /* calculate the new offset */
                }
                else {
                        if (i % 4) {                            /* fill section 2 */
                                outp (922, xmsr_on);            /* ensure we are writing to section 2 or 3 */
                                dspln (offset, (unsigned) 40960, adrs1);
                        }
                        else {                                  /* fill section 0 */
                                outp (922, xmsr_off);           /* ensure we are writing to section 0 or 1 */
                                dspln (offset, (unsigned) 40960, adrs1);
                        }
                        adrs1 += 320L;                          /* calculate the new offset */
                }
        return (0);                                             /* return to caller */
}

/*****  name:  enhance (&buffer[])
  This routine provides the enhancement (or color scale) curve.  The address of
a line of data is provided on input.
global variables used:
        int scale[]                     ;color scale
routine callers:
        main()
*/
enhance (buffer)
char buffer[];                          /* one line of data */
{
        unsigned int i;                         /* general counter */

        for (i = 0; i < 64000; i++)
                buffer[i] = scale[buffer[i]];
        return(0);                              /* normal return */
}

/*****  name:  erase (row, col)
This routine writes 40 spaces to the display starting at the row, column.
global variables used:
        none
routine callers:
        anyone()
*/
erase (row, col)
int row;
int col;
```

```
{
        char *ptr;

        ptr = "                                                ";       /* define space string */
        display (row, col, 7, ptr);                                     /* display string */
        return(0);
}

/*****  name:  menu()
This routine displays the main menu.
global variables used:
        struct page pag0[]              ;displayed data structure
routine callers:
        main()
*/
menu()
{
        int i;                                  /* general counter */

        clear();                                        /* clear the screen */
        for (i = 0; pag0[i].row >= 0; i++)              /* display the menu */
                display (pag0[i].row, pag0[i].col, pag0[i].attrib, pag0[i].line);
        return(0);
}

/*****  name:  onepix()
This routine displays a single satellite picture in either high resolution
(640x400x16), or low resolution (640x200x16).
global variables used:
        struct page pag1[]                      ;display data structure
routine callers:
        main()
*/
onepix()
{
        int i;                                  /* general counter */
        int fd;                                 /* file descriptor */
        int resoltn;                            /* resoltn of displayed pix */
        int lines;                              /* number of lines to display */
        int temp;                               /* temporary variable */
        long size;                              /* size of memory data buffer */
        char *adrs;                             /* memory buffer pointer */
        char s[15];                             /* general string variable */

        /* display menu for this option */
        clear();                                                /* clear the screen */
        for (i = 0; pag1[i].row >= 0; i++)
                display (pag1[i].row, pag1[i].col, pag1[i].attrib, pag1[i].line);

        /* retrieve the desired resolution */
        display (4,10,10,"Please enter 0 or 1:  ");     /* prompt the user */
        if (getstr (&s[0], 1))                          /* get user input */
```

```
        return(0);                              /* return on ESC key */
resoltn = s[0] - '0';                           /* convert to decimal */
if (resoltn > 1)                                /* response in valid range? */
        resoltn = 1;                            /* let user's mistake be high resolution */

/* get the data filename */
display (9,10,10,"Enter the satellite data filename [");
display (9,60,10,"]");
for (;;) {                                      /* filename entry loop */
        display (9,45,7,"");                    /* position cursor */
        if (getstr (&s[0], 14))                 /* get user input */
                return(0);
        if ((fd = Dopen (&s[0], O_RDONLY)) == ERROR) {  /* open the file */
                display (24,24,207," Data file not found, try again ");
                continue;
        }
        break;                          /* exit loop */
}

/* display section */
erase (24,24);                                  /* erase any error msg that is present */
if (resoltn)                                    /* determine the number of lines to display */
        lines = hirsltn;                        /* high resolution */
else
        lines = lorsltn;                        /* low resolution */
cursor (0);                                     /* turn the cursor off */
display (15, 28, 138, "****  PLEASE WAIT  ****");       /* have patience */
display (17, 23, 10, "Satellite data is being processed.");
if (pixload (fd, &adrs, &size, lines)) {                /* load the data into memory */
        Dclose (fd);                            /* error occurred - close the file */
        display (24, 25, 207, "        File read error       ");
        delay (73);                             /* let user read error msg */
        cursor (1);                             /* turn the cursor back on */
        return(0);                              /* return to caller */
}
if (resoltn) {                                  /* setup the graphics mode */
        mode (&pixdsp[0]);
        highdsp (adrs);
}
else {
        mode (&pixloop1[0]);                    /* low resolution */
        lowdsp (adrs);                          /* display the picture */
}

/* wait for an ESC key */
for (i = 1; i; ) {
        while (!(temp = getchr()));             /* wait for any keyboard input */
        if ((temp & 255) == 27)                 /* an ESC key? */
                i = 0;                          /* yes - exit loop */
}
Dclose (fd);                                    /* close the input file */
rbrk();                                         /* release the memory buffer */
```

```
        mode (&alpha[0]);                                   /* change back to alphanumeric mode */
        return(0);
}

/*****  name:  loopix()
This routine loops a series of pictures at low resolution (640x200x16).
global variables used:
        struct page pag2[]
routine callers:
        main()
*/
loopix()
{
        int i;                          /* general counter */
        int fd;                         /* file descriptor */
        int row;                        /* display row value */
        int cflag;                      /* continue on flag */
        int ticks, tym;                 /* timer variables */
        int sectff;                     /* section flip-flop flag */
        int temp;                       /* temporary variable */
        int xmsr;                       /* extended mode register value */
        char c;                         /* general purpose char variable */
        char s[15];                     /* input char string */
        struct pixfile {                /* struct for tracking data in memory */
                char *adrs;             /* picture data addresses */
                long size;              /* size of the data */
        } pixs[18];

        /* display instructional lines */
        clear();
        for (i = 0; pag2[i].line[0]; i++)
                display (pag2[i].row, pag2[i].col, pag2[i].attrib, pag2[i].line);

        /* prompt the user for input filenames */
        for (row = 2, i = 0, cflag = 1; row < 18 && cflag;) {
                display (row, 39, 2, "]");
                display (row, 3, 2,"Enter data filename [");            /* display input prompt */
                if (getstr (&s[0], 14)) {                               /* get a filename */
                        pixs[i].size = 0;                               /* terminate the structure */
                        cflag = 0;                                      /* remove any pictures already here */
                        break;
                }
                if (s[0] == 0) {                                        /* entries finished? */
                        pixs[i].size = 0;                               /* terminate the loop */
                        break;
                }
                if ((fd = Dopen (&s[0], O_RDONLY)) == ERROR) {          /* open the data file */
                        display (24, 25, 207, "Data file not found, try again");
                        continue;
                }

                /* process the data into a memory buffer */
```

```
        cursor (0);                                             /* turn the cursor off */
        erase (24,25);                                          /* erase any error msg */
        display (6, 48, 138, "*****  PLEASE WAIT  *****");
        display (9, 43, 10, "Satellite data is being processed.");
        if (pixload (fd, &pixs[i].adrs, &pixs[i].size, lorsltn)) {      /* error occurred? */
                Dclose (fd);
                display (24, 25, 207, "        File read error       ");
                delay (73);                                     /* let user read error message */
                cflag = 0;                                      /* errors here are fatal to loop */
                cursor (1);                                     /* be sure the cursor is on */
                break;
        }
        Dclose (fd);
        erase (6,40);                                           /* clear off all screen msgs */
        erase (9,40);
        erase (24,25);                                          /* remove any error msgs */
        cursor (1);                                             /* turn the cursor back on */

        if (pixs[i].size == 0) {                                /* out of memory? */
                display (6, 52, 10, "MEMORY EXHAUSTED");
                display (9, 43, 10, "Your loop will start in 5 seconds.");
                delay (91);
                break;
        }
        i++;                                                    /* maintain the proper indices */
        row++;
}

/* data processing looping loop */
while (cflag) {                                                 /* ensure one picture available */
        mode (&pixloop1[0]);                                    /* switch into graphics mode */
        lowdsp (pixs[0].adrs);                                  /* display the first picture */
        xmsr = pixloop1[15];                                    /* init XMSR reg value */
        sectff = 0;                                             /* init memory section flip-flop */
        for (tym = 3, i = 1; cflag; i++) {
                xmsr ^= 64;                                     /* flip the XMSR RAM fill bit 6 */
                outp (922, xmsr);                               /* switch graphic memory input to the other
                ticks = tym * 4 + 1;                            /* calculate the number of ticks; 1,5,9,13,

                timer (&ticks);                                 /* set the timer going */
                if (pixs[i].size == 0) {                        /* at the end of loop? */
                        i = 0;                                  /* yes - reset to beginning */
                        ticks += 9;                             /* add 0.5 secs to the delay at the end */
                }
                lowdsp (pixs[i].adrs);                          /* put the next picture in the graphics car
                while ((ticks && cflag) || (temp = getchr())) { /* wait for the timer to expire */
                        c = (temp & 65280) >> 8;                /* yes - isolate key code */
                        switch (c) {                            /* execute proper key functions */
                            case 1:                             /* the ESC key */
                                cflag = 0;                      /* terminate the loop */
                                break;
                            case 80:                            /* down arrow key, slow down */
```

```
                                    if (tym < 6)                    /* only six time levels of looping */
                                        tym += 1;                   /* change the time interval */
                                    break;
                                case 72:                            /* up arrow, speed up */
                                    if (tym > 0)
                                        tym -= 1;                   /* decrease the time interval */
                                    break;
                                case 57:                            /* space bar, stop the picture */
                                    for(;;) {                       /* this could take awhile, maybe even forev
                                        while (!(temp = getchr())); /* wait for keyboard entry */
                                        c = (temp & 65280) >> 8;    /* isolate the key code again */
                                        if (c == 1) {               /* the dreaded panic key strikes */
                                            cflag = 0;              /* end the loop */
                                            break;                  /* exit forever loop */
                                        }
                                        else if (c == 57)           /* the only other useful key here is the sp
                                            break;                  /* resume the loop */
                                    }
                                    break;
                                default:                            /* anything else just loops */
                                    break;
                            }
                        }
                        if (sectff) {
                                xmsr = pixloop1[15];                /* first section */
                                sectff--;
                        }
                        else {
                                xmsr = pixloop2[15];                /* second section */
                                sectff++;
                        }
                        outp (922, xmsr);                           /* display the next picture */
                }
        }
        rbrk();                                                     /* deallocate memory */
        mode (&alpha[0]);
        return(0);                                                  /* normal return */
}

/*****  name:  colors()
This routine changes the color scale used to display the satellite pictures.
global variables used:
        struct page pag3[]                      ;display data
        char scale[]                            ;color scale
routine callers:
        main()
*/
colors()
{
        int i, j;                       /* general counter and display indice */
        int fd;                         /* file descriptor */
        int row;                        /* current row value in editor */
```

```
int cflag;                      /* continue on flag */
int temp1, temp2;               /* temporary variables */
char edcolor;                   /* current color selection for editor */
char tscale[16];                /* temporary color curve */
char s[15];                     /* general purpose string */
char c;                         /* general character variable */

/* display first portion of screen */
clear();                                        /* clear the screen */
for (i = 0; pag3[i].line[0]; i++)
        display (pag3[i].row, pag3[i].col, pag3[i].attrib, pag3[i].line);

/* let the user respond */
if (getstr (&s[0], 1))
        return(0);                              /* return on the ESC key */
if (toupper(s[0]) == 'Y') {                     /* load in a previous curve from the diskette */
        display (5,3,10,"Enter the color curve diskette");
        display (6,3,10,"filename [");
        display (6,27,10,"]");
        for(;;) {
                display (6,13,7,"");            /* position cursor */
                if (getstr (&s[0], 14))         /* get user's response */
                        break;
                if ((fd = Dopen (&s[0], O_RDONLY)) == ERROR) {  /* open the file */
                        display (24,27,207," File not found, try again ");
                        continue;               /* reenter loop */
                }
                if (Dread (fd, &scale[0], 256) == ERROR) {
                        display (24,27,207,"      File read error      ");
                        delay (73);             /* delay 4 secs so user can read error msg */
                }
                Dclose (fd);                    /* close file */
                break;
        }
        return (0);
}

/* complete the display for editing */
for (i++; pag3[i].line[0]; i++)
        display (pag3[i].row, pag3[i].col, pag3[i].attrib, pag3[i].line);
for (i = 0; i < 16; i++)                        /* setup the temporary scale */
        tscale[i] = (scale[i] & 15);            /* use only the right nibble */
dspscale (3, 51, &tscale[0]);
dspscale (3, 73, &tscale[0]);                   /* display temporary scale */

/* enable the editing to take place */
row = 3;                                        /* init the display row */
edcolor = 0;                                    /* set the current editing color */
arrow (row, edcolor);                           /* place an arrow on level 0 */
cursor (0);                                     /* turn the cursor off */
for (cflag = 1; cflag;) {                       /* edit processing loop */
        while (!(c = ((getchr() & 65280) >> 8)));       /* wait for entry, and use the key code */
```

```
switch (c) {                                    /* process the key stroke */
    case 1:                                 /* ESC key */
        cflag = 0;                              /* return to caller */
        break;
    case 77:                                /* right arrow key */
        edcolor++;                              /* inc editing color code */
        if (edcolor > 15)                       /* circle around if past white */
                edcolor = 0;
        arrow (row, edcolor);                   /* put new color on display */
        break;                                  /* reenter loop */
     case 75:                               /* left arrow key */
        if (edcolor == 0)                       /* cycle when past black */
                edcolor = 15;
        else
                edcolor--;                      /* decrement color */
        arrow (row, edcolor);                   /* put the new color on the screen */
        break;                                  /* reenter loop */
    case 72:                                /* up arrow key */
        temp1 = row;                            /* save the current row */
        row--;                                  /* adjust row value */
        if (row < 3) {                          /* check for past top of scale */
                row++;                          /* correct the row value */
                printf("^G");                   /* hopefully the user is not deaf too */
        }
        if (temp1 != 19)                        /* don't rewrite the last row */
                unarrow (temp1, tscale[temp1 - 3]);
        else
                unarrow (temp1, 0);             /* put black next to 'update' */
        arrow (row, edcolor);                   /* move arrow to new location */
        break;                                  /* reenter loop */
    case 80:                                /* down arrow key */
        temp1 = row;                            /* save the current row */
        row++;                                  /* adjust row value */
        if (row > 19) {                         /* can't go past 'update scale' */
                row--;                          /* correct row value */
                printf("^G");                   /* beep user */
        }
        unarrow (temp1, tscale[temp1 - 3]);     /* erase the current arrow */
        if (row == 19)                          /* no color if on 'update' */
                arrow (row, 0);
        else                                    /* place new color/arrow on display */
                arrow (row, edcolor);
        break;                                  /* reenter loop */
    case 28:                                /* carriage return, change the scale */
    case 57:                                /* space bar, change the scale */
        if (row != 19) {                        /* not updating the scale */
                tscale[row - 3] = edcolor;      /* place in the editing color */
                break;                          /* reenter loop */
        }

        /* update the 'real' color curve */
        for (i = 0; i < 16; i++) {              /* expand the scale to two bytes */
```

```
                temp1 = i << 4;                 /* get the starting row */
                temp2 = tscale[i] << 4;         /* get the starting value */
                for (j = 0; j < 16; j++)
                        scale[temp1 + j] = temp2 | tscale[j];
        }
        dspscale (3, 51, &tscale[0]);           /* display the new current scale */

        /* save new scale for future use? */
        cursor (1);                             /* turn the cursor back on */
        display (5,3,10,"Do you want to save the new color");
        display (6,3,10,"scale on diskette (Y/N)?  ");
        if (getstr (&s[0], 1)) {                /* get the user's response */
                cflag = 0;                      /* exit loop */
                break;
        }
        if (toupper(s[0]) != 'Y') {             /* only a Y can save file */
                cflag = 0;                      /* exit loop */
                break;
        }
        display (8,3,10,"Enter the filename [");
        display (8,37,10,"]");
        for(;;) {                                       /* get the filename */
                display (8,23,7,"");                    /* position cursor */
                if (getstr (&s[0], 14))                 /* wait for entry */
                    break;
                if ((fd = Dopen (&s[0], O_WRONLY)) != ERROR) {
                    display (24,29,207," File already exists ");
                    display (10,3,10,"Overwrite previous file (Y/N)?  ");
                    if (getstr (&s[0], 14))             /* get user response */
                        break;
                    if (toupper(s[0]) != 'Y') {         /* don't overwrite */
                        erase (10,0);                   /* clear error msgs */
                        erase (24,20);
                        display (6,13,7,"              ");
                        continue;                       /* get another filename */
                    }
                }
                else
                    fd = Dcreat (&s[0], 0);     /* create a new file */
                if (Dwrite (fd, &scale[0], 256) == ERROR) {
                    display (24,29,207,"   File write error    ");
                    delay (73);                 /* let user read error msg */
                }
                Dclose (fd);                    /* close file */
                break;
        }
        cflag = 0;                              /* exit the loop */
        break;
    default:                            /* every other key stoke */
        printf("^G");                           /* beep for errors */
        break;
}
```

```
        }
        cursor (1);                                          /* ensure cursor is on */
        return(0);                                           /* normal return */
}

/*****  name:  arrow (row, color)
This routine displays an arrow and the editing color for colors().
global variables used:
        none
routine callers:
        colors()
*/
arrow (row, color)
int row;                                /* row to display arrow on */
char color;                             /* color of block arrow points towards */
{
        display (row, 73, color,"■■");          /* display color block */
        display (row, 76, 10, "^[-");           /* display the arrow */
        return(0);
}
/*****  name:  unarrow (row, color)
This routine removes the arrow from a row, and restores the proper color.
global variables used:
        none
routine callers:
        colors()
*/
unarrow (row, color)
int row;
char color;
{
        display (row, 73, color, "■■");         /* restore the proper color */
        display (row, 76, 7, "  ");             /* remove the arrow */
        return(0);
}
/*****  name:  dspscale (row, col, scale)
This routine displays the color scale(s) for editing.
global variables used:
        none
routine callers:
        colors()
*/
dspscale (row, col, scale)
int row;                                        /* row to start scale on */
int col;                                        /* column to start scale on */
char scale[];                                   /* scale to display */
{
        int i;                                  /* general counter */

        for (i = 0; i <16; i++, row++)
                display (row, col, scale[i], "■■");
        return(0);
```

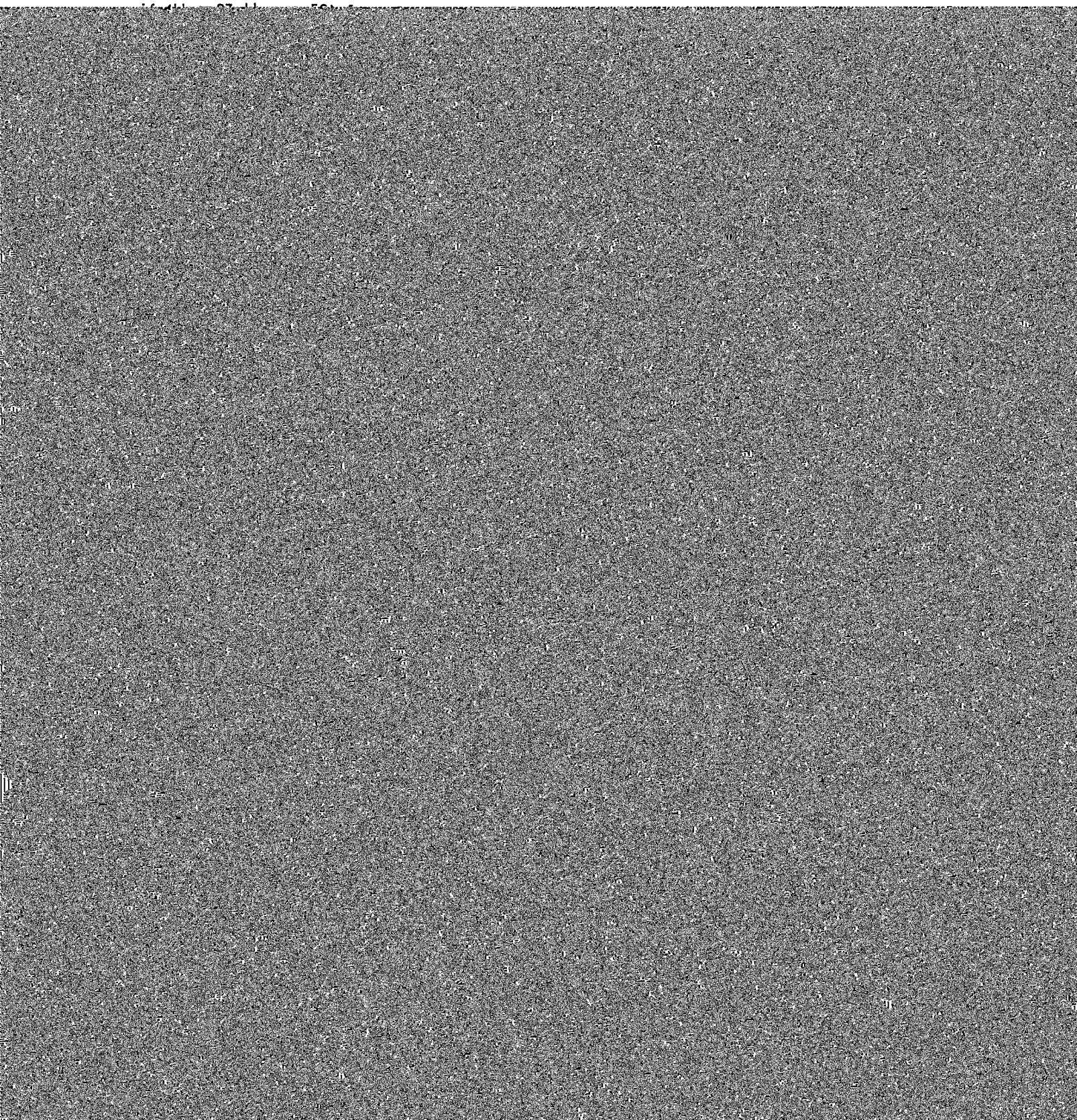
```
}
/*****  name:  cursor (off_on)
This routine turns the cursor on and off (off_on = 1 is on, off_on = 0 is off).
global variables used:
        char alpha[]                    ;6845 register values
routine callers:
        colors()        loopix()
*/
cursor (off_on)
int off_on;                             /* on/off flag */
{
        int c;                                  /* character variable */

        if (off_on)
                c = alpha[3];                   /* turn off bit 5 of R10 */
        else
                c = alpha[3] | 32;              /* turn on bit 5 of R10 */
        outp (916, 10);                         /* set pointer reg to 10 */
        outp (917, c);                          /* set the cursor */
        return (0);
}
```

3. Wait for the Desired Time (WAIT)

The wait program waits until the specified input time and returns control to DOS for the execution of another program. This routine was designed to be implemented in a batch file for the automation of satellite data retrieval.

```
/*      date:  May 1987
        version:  1.0
        by:  Glen W. Sampson, NWS WRH SSD

Purpose:
  This program is designed to act as a timer by waiting for the command
line input time to occur.

Exits:
  The only routine exit is for the input time to occur or a Ctrl C command
from the keyboard.

Created by:
        (version 3.00 Lattice C)
        C>lc -cu -md wait
        C>link cd wait func, wait, con, lcd
*/
/* global variables used in func.asm */
int prodflg;
int optr;
int tx_ptr;

void main (argc, argv)
int argc;                               /* number of command line arguments */
char **argv;                            /* pointer to the arguments */
{
        struct hhmm {
                char mins;                      /* current minutes */
                char hour;                      /* current hour */
        } x;
        int i;                                  /* general variable */
        char hh, mm;                            /* input hours and minutes */
        char s[10];                             /* general string variable */

        /* process the command line */
        if (argc != 2) {
                printf("Invalid the command line^G\n");
                printf("USAGE:  wait hh:mm\n");
                exit();
        }

        /* break the input time apart */
        for (i = 0; *argv[1] != ':'; i++)       /* grab the hours */
```

```
        s[i] = *argv[1]++;
s[i] = 0;                       /* terminate the string */
hh = atoi (&s[0]);              /* convert to decimal */
argv[1]++;                      /* skip over the colon */
for (i = 0; *argv[1]; i++)      /* grab the minutes */
        s[i] = *argv[1]++;
s[i] = 0;
mm = atoi (&s[0]);
```



4. Assembly Language Functions (FUNC and AOS)

Assembly language usage has been avoided wherever possible in the IBM PC portion of the software. Most of the assembly language functions manipulate the hardware directly, or provide general interfaces to MS-DOS not found in the C compiler libraries. The functions contained in the FUNC.ASM file are:

timer	getchr	putchr	display
dspln	clear	reset	setcom
set_td	tx_chk	satdgtl.	

Only one function is contained in the AOS.ASM file called setaos.

```
;       date:  March 1987
;       version:  4.1 (Satellite Project version)
;       by:  Glen W. Sampson, NWS WRH SSD
;
;Purpose:
;  These routines contain BIOS functions calls not normally available
;in C, and device drivers for the asynchronous communication adapters.
;
;Exits:
;  No exits exist from these routines.
;
;Created by:
;       (Version 1.0 IBM Macro Assembler)
;       C>masm func, func, nul, nul
;
;*************************************************************************
; setup the proper memory model
S_MODEL         EQU     0
P_MODEL         EQU     0
D_MODEL         EQU     1
L_MODEL         EQU     0
;
;*************************************************************************
; setup the stack offsets
        IF      L_MODEL OR P_MODEL              ;FAR CALLS
FIRST_PARAM     EQU     6
SECOND_PARAM    EQU     8
THIRD_PARAM     EQU     10
FOURTH_PARAM    EQU     12
FIFTH_PARAM     EQU     14
SIXTH_PARAM     EQU     16
        ELSE
FIRST_PARAM     EQU     4                       ;NEAR CALLS
SECOND_PARAM    EQU     6
THIRD_PARAM     EQU     8
```

```
FOURTH_PARAM    EQU     10
FIFTH_PARAM     EQU     12
SIXTH_PARAM     EQU     14
        ENDIF
;
; ****************************************************************
; declare external functions (if appropriate)
        IF P_MODEL OR L_MODEL
;       EXTRN   FUNCTION:FAR
        ENDIF
;
; ****************************************************************
; establish the DATA SEGMENT
DGROUP  GROUP   DATA
DATA    SEGMENT WORD    PUBLIC  'DATA'
        ASSUME  DS:DGROUP
;
; Function variables
;
;       A/D converter variables
        BUF_PTR         DW      00H             ;buffer pointer
        SBUF_OFST       DW      00H             ;start of buffer defined in C
        BUF_SEG         DW      00H             ;SEG of buffer
        EBUF_OFST       DW      00H             ;end of buffer
        STORFLG         DW      00H             ;to store or not to, this is the flag
        LINENUM         DW      00H             ;current line number we are sampling
        PIXELS          DW      00H             ;number of samples yet to do in our current scan
;
;       timer interrupt variables
        TYMOFST         DW      00H             ;timer ticks adrs offset
        TYMSEG          DW      00H             ;timer ticks adrs seg
        DUMOFST         DW      00H             ;dummy routine to handle timer break int
        DUMSEG          DW      00H             ;dummy routine segment adrs
;
;       asynchronous function variables
;               receive
        EXTRN           PRODFLG:WORD            ;product received flag
        BUFO            DW      00H             ;start of the receipt buffer
        BUFOMAX         DW      00H             ;end of the receipt buffer
        BUFOSEG         DW      00H             ;segment which contains the receipt buffer
        PTRO            DW      00H             ;current buffer pointer
        BEGINO          DW      00H             ;offset for the start of product
;
;               transmit
        EXTRN           TX_PTR:WORD             ;transmit pointer
        EXTRN           OPTR:WORD               ;output buffer pointer
        TD_SEG          DW      00H             ;transmit buffer segment
        TD_OFST         DW      00H             ;transmit buffer offset
;
;       continuous speech variables
        TXOFST          DW      00H             ;control buffer structure offset
        TXSEG           DW      00H             ;control buffer structure segment
```

```
        SILOFST         DW      00H                     ;silence buffer offset
        SILSEG          DW      00H                     ;silence buffer segment
DATA    ENDS                                            ;end data segment
;
;*********************************************************************
; establish the GROUP and SEGMENT definitions
        IF      S_MODEL
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE    PUBLIC  'PROG'
        ASSUME  CS:PGROUP
        ENDIF

        IF      P_MODEL
PGROUP  GROUP   CODE
_CODE   SEGMENT BYTE    PUBLIC  '_CODE'
        ASSUME  CS:PGROUP
        ENDIF

        IF      D_MODEL
CGROUP  GROUP   CODE
CODE    SEGMENT BYTE    PUBLIC  'CODE'
        ASSUME  CS:CGROUP
        ENDIF

        IF      L_MODEL
CGROUP  GROUP   _PROG
_PROG   SEGMENT BYTE    PUBLIC  '_PROG'
        ASSUME  CS:CGROUP
        ENDIF

;
;*********************************************************************
; put the hooks in for 'C' access
;       PUBLIC  TIME                    ;functions in this source code
;       PUBLIC  DATE
;       PUBLIC  DELETE
        PUBLIC  TIMER
        PUBLIC  GETCHR
        PUBLIC  PUTCHR
        PUBLIC  DISPLAY
        PUBLIC  DSPLN
        PUBLIC  CLEAR
;       PUBLIC  SEARCH
        PUBLIC  RESET
        PUBLIC  SETCOM
;       PUBLIC  SETREAD
        PUBLIC  SET_TD
;       PUBLIC  TX_CONT
        PUBLIC  TX_CHK
        PUBLIC  SATDGTL
          PUBLIC  LINENUM                       ;line number variable
          PUBLIC  BUF_PTR
```

```
;
;
;*************************************************************************
;
;                       f u n c t i o n s
;
;*************************************************************************
;
;       name:  time(adrs)
;       version:  1.0
;  This routine retrieves a 2 byte value containing the hour and
;minutes.  These values are stored at input address.
;
        IF      S_MODEL OR D_MODEL
TIME_PROC       PROC    NEAR
        ELSE
TIME_PROC       PROC    FAR
        ENDIF

TIME:
        PUSH    BP                      ;STACK'EM UP
        MOV     BP,SP
        PUSH    DS
        MOV     AH,2CH                  ;GET TIME FUNCTION CALL
        INT     21H                     ;DOS CAN DO IT
        MOV     BX,FIRST_PARAM[BP]      ;GET OFFSET
    IF  L_MODEL OR D_MODEL
        MOV     AX,SECOND_PARAM[BP]     ;RETRIEVE SEGMENT ADRS
        MOV     DS,AX                   ;SETUP THE SEGMENT
    ENDIF
        MOV     DS:[BX],CX              ;STORE IN INPUT ADRS
        MOV     AL,CH                   ;PUT THE HOUR INTO 'A'
        MOV     BL,64H                  ;LOAD UP MULT OPERAND
        MUL     BL                      ;MULTIPLY BY 100
        AND     CX,0FFH                 ;ISOLATE THE MINUTES
        ADD     AX,CX                   ;COMBINE HOUR & MIN INTO INT
        POP     DS                      ;MOVE'EM OUT
        POP     BP
        RET                             ;RETURN TO CALLER
TIME_PROC       ENDP
;
;       name:  date (&buf)
;  This routine retrieves the current data from the system and places the values
;int the input buffer address.
;
        IF S_MODEL OR D_MODEL
DATE_PROC       PROC    NEAR
        ELSE
DATE_PROC       PROC    FAR
        ENDIF

DATE:
```

```
        PUSH    BP                      ;STACK'EM UP
        MOV     BP,SP
        PUSH    DS
        MOV     AH,2AH                  ;GET THE DOS CALL CODE
        INT     21H                     ;CALL DOS
        MOV     BX,FIRST_PARAM[BP]      ;RETRIEVE THE BUFFER OFFSET ADRS
      IF L_MODEL OR D_MODEL
        MOV     AX,SECOND_PARAM[BP]     ;GET THE SEG ADRS
        MOV     DS,AX
      ENDIF
        MOV     [BX],CX                 ;PUT YEAR IN THE BUFFER
        INC     BX                      ;UPDATE THE OFFSET
        INC     BX
        MOV     [BX],DX                 ;PLACE THE MONTH AND DAY IN THE BUFFER
        POP     DS                      ;RETURN TO CALLER
        POP     BP
        RET
DATE_PROC       ENDP
;
;       name:  set_td(&buffer)
;       version:  1.0
;  This routine transmits a buffer of data out of an async. port using interrupts
;to signal when the next character can be transmitted.  This routine allows a
;program to start the transmission and then execute other tasks while the data
;is being sent out.  This routine can not be used in conjunction with the
;interrupt receive routine unless two async. cards are present.

        IF S_MODEL OR D_MODEL
SET_TD_PROC     PROC    NEAR
        ELSE
SET_TD_PROC     PROC    FAR
        ENDIF

; define the base port adrs
        TD_BASE         EQU     3F8H            ;async. base port adrs
        TD_IRQ          EQU     4               ;type of interrupt
SET_TD:
        PUSH    BP                              ;STACK'EM UP
        MOV     BP,SP
        PUSH    ES
        PUSH    DI
        MOV     AX,FIRST_PARAM[BP]              ;GET THE BUFFER ADRS OFFSET
        MOV     TD_OFST,AX                      ;SAVE OFFSET IN MEMORY
      IF D_MODEL OR L_MODEL
        MOV     AX,SECOND_PARAM[BP]             ;GET THE SEG ADRS
      ELSE
        MOV     AX,DS                           ;USE THE CURRENT SEGMENT
      ENDIF
        MOV     TD_SEG,AX                       ;SAVE THE SEG ADRS IN MEMORY
;
;  INITIALIZE THE INTERRUPT CONTROLLER (8259) AND TABLE
        XOR     AX,AX                           ;CLEAR AX
```

```
        MOV     ES,AX                           ;POINT THE SEG TO TABLE
        MOV     DI,(TD_IRQ + 8) * 4             ;GET THE TABLE OFFSET
        MOV     AX,OFFSET TD_INT                ;GRAB THE ISR OFFSET
        CLD                                     ;STRING FORWARD DIRECTION
        STOSW                                   ;STORE THE OFFSET
        MOV     AX,CS                           ;GET THE CURRENT CODE SEGMENT
        STOSW                                   ;STORE THE CODE SEG IN TABLE
        IN      AL,21H                          ;GET THE CURRENT 8259 STATUS
      IF TD_IRQ EQ 4
        AND     AL,0EFH                         ;PRIMARY PORT
      ELSE
        AND     AL,0F7H                         ;SECONDARY PORT
      ENDIF
        CLI                                     ;DISABLE ANY INTERRUPTIONS
        OUT     21H,AL                          ;ENABLE ASYNC INTS
        STI                                     ;ALLOW INTERRUPTIONS
;
;  SETUP THE ASYNC. CARD FOR EASY INTERRUPT ENABLES
        MOV     DX,PORT                         ;GET THE BASE ADRS
        ADD     DX,3                            ;POINT TO THE LINE CONTROL REG
        IN      AL,DX                           ;GET THE CURRENT VALUE
        AND     AL,07FH                         ;DLAB = 0
        OUT     DX,AL                           ;ALLOW INT TO BE CHGED
        SUB     DX,2                            ;POINT TO THE INT ENABLE REG
        XOR     AL,AL
        OUT     DX,AL                           ;REMOVE ANY INTS
        ADD     DX,3                            ;POINT TO MODEM CTRL REG
        MOV     AL,08H                          ;SET OUT2 BIT
        OUT     DX,AL                           ;ALLOW INTS, SHUD BE HIGH AT RESET THO
;
        POP     DI                              ;RETURN TO CALLER
        POP     ES
        POP     BP
        RET
;
;  TRANSMIT INTERRUPT SERVICE ROUTINE
;  REGISTERS USED:
;       AX - VARIABLE
;       BX - CURRENT BUFFER TRANSMIT POINTER
;       CX - CURRENT BUFFER FILL POINTER
;       DX - PORT BASE ADRS
;       ES - OUTPUT BUFFER SEGMENT
;       DS - POINTER SEGMENT
TD_INT:
        PUSH    DS                              ;PRESERVE CURRENT OPERATIONS
        PUSH    ES
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
;
;  SETUP THE REGS TO DO SOME WORK
```

```
        MOV     AX,SEG TD_SEG                   ;GET THE DATA SEG
        MOV     DS,AX
        MOV     AX,TD_SEG                       ;GET THE BUFFER SEG
        MOV     ES,AX                           ;PUT THE OBUF[] SEG IN ES
        MOV     AX,TD_OFST                      ;GET THE OBUF[] OFFSET
        MOV     DX,AX

        MOV     AX,SEG TX_PTR                   ;GET THE POINTER SEG
                                                ;LIKELY THE SAME AS DATA SEG
        MOV     DS,AX
        MOV     BX,TX_PTR                       ;GET THE CURRENT TX PTR
        MOV     AX,OPTR                         ;GET THE CURRENT DATA FILL POINTER
        MOV     CX,AX                           ;SAVE FILL POINTER IN CX
;
;  DETERMINE THE PROPER ACTION TO TAKE
        CMP     BX,CX                           ;ANY DATA AVAILABLE?
        JZ      TD_STOP                         ;NO - DISABLE ASYNC INT
        ADD     BX,DX                           ;YES - DETERMINT OFFSET TO NEXT BYTE
        MOV     AL,ES:[BX]                      ;GRAB A BYTE, BUT DON'T EAT IT!!
        INC     TX_PTR                          ;UPDATE THE TX PTR
        MOV     DX,TD_BASE                      ;SETUP PORT BASE
        JMP     TD_RTN                          ;SERVICE AND RETURN
TD_STOP:
        XOR     AL,AL                           ;CLEAR AL
        MOV     DX,TD_BASE                      ;SETUP PORT BASE
        INC     DX                              ;POINT TO INT ENABLE REG
TD_RTN:
        OUT     DX,AL                           ;RESPOND TO THE ASYNC CARD
        MOV     AL,20H                          ;CLEAR THE 8259
        CLI                                     ;ENSURE INT ARE OFF
        OUT     20H,AL                          ;CLEAR THE INT
        STI                                     ;ENABLE INT
        POP     DX                              ;RESTORE ORDER
        POP     CX
        POP     BX
        POP     AX
        POP     ES
        POP     DS
        IRET
SET_TD_PROC     ENDP

;
;       name:  delete(&filename)
;  DOS delete a file                                                    *
;
        IF S_MODEL OR D_MODEL
DELETE_PROC     PROC    NEAR
        ELSE
DELETE_PROC     PROC    FAR
        ENDIF

DELETE:
```

```
        PUSH    BP                      ;stack 'em up
        MOV     BP, SP
        PUSH    SI
        PUSH    DI
        PUSH    DS
        MOV     DX, FIRST_PARAM[BP]     ;offset
      IF D_MODEL OR L_MODEL
        MOV     AX, SECOND_PARAM[BP]    ;segment
        MOV     DS, AX
      ENDIF
        MOV     AH, 41H
        INT     21H
        POP     DS
        POP     DI
        POP     SI
        POP     BP
        RET
DELETE_PROC     ENDP
;
;       name:  timer (ticks value adrs)
;       version:  1.0
;  This routine uses the timer channel 0 interrupts to delay for a period of time.
;The greatest error in the delay is 0.05495 seconds due to the interrupt occurance
;of 18.2 per second.  The address of the number of ticks to delay is given upon
;entry to this routine.  One tick equals 0.05495 seconds; you can delay a maximum
;of 65536 ticks, or about 3600 seconds (1 hour).
;
        IF      S_MODEL OR D_MODEL
TIMER_PROC      PROC    NEAR
        ELSE
TIMER_PROC      PROC    FAR
        ENDIF

; DEFINE THE SOFTWARE TIMER BREAK SOFTWARE INT VECTOR (FROM BIOS LISTINGS)
        TYMINT  EQU     1CH             ;BREAK ADRS IN TIMER_INT BIOS ROUTINE

TIMER:
        PUSH    BP                      ;STACK'EM UP
        MOV     BP,SP
        PUSH    ES
        PUSH    DI
        MOV     AX,FIRST_PARAM[BP]      ;GET THE TYMFLG OFFSET VALUE
        MOV     TYMOFST, AX             ;SAVE THE OFFSET ADRS IN MEMORY
      IF D_MODEL OR L_MODEL
        MOV     AX,SECOND_PARAM[BP]     ;GET THE TYMFLG SEGMENT VALUE
      ELSE
        MOV     AX, DS                  ;USE THE CURRENT DATA SEGMENT
      ENDIF
        MOV     TYMSEG, AX              ;SAVE THE SEGMENT ADRS IN MEMORY
;
; RETRIEVE THE CURRENT DUMMY RETURN ADRS IN TIMER BREAK INT
        XOR     AX,AX                   ;CLEAR THE AX REGISTER
```

```
        MOV     ES,AX                   ;SET ES FOR VECTOR INT TABLE SEGMENT
        MOV     BX,TYMINT * 4           ;GET THE TIMER BREAK ADRS OFFSET
        MOV     AX,ES:[BX]              ;RETRIEVE THE OFFSET ADRS
        MOV     DUMOFST,AX              ;SAVE DUMMY OFFSET ADRS IN MEMORY
        INC     BX                      ;SET BX POINTER TO SEGMENT ADRS
        INC     BX
        MOV     AX,ES:[BX]              ;RETRIEVE THE SEGMENT ADRS
        MOV     DUMSEG,AX               ;SAVE THE SEGMENT ADRS IN MEMORY
;
; ESTABLISH THE NEW TIMER BREAK ADRS
        MOV     DI,TYMINT * 4           ;SETUP THE INDEX REGISTER
        MOV     AX,OFFSET TYM_INT       ;GET THE ISR ADRS OFFSET
        CLD                             ;STORE FORWARD DIRECTION
        CLI                             ;DISABLE ANY INT WHILE WE DO THIS
        STOSW                           ;STORE OFFSET ADRS
        MOV     AX,CS                   ;GET THE CURRENT CODE SEGMENT
        STOSW                           ;STORE THE NEW CODE SEGMENT
        STI                             ;OFF WE GO - ENABLE INTERRUPTS
        POP     DI                      ;CLEAR OFF THE STACK
        POP     ES
        POP     BP
        RET                             ;RETURN TO CALLER
;
; TIMER BREAK ADRS INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
;       AX - VARIABLE, BUT GENERALLY THE CURRENT TICKS VALUE
;       BX - VARIABLE, BUT GENERALLY OFFSET OF TICKS VALUE ADRS
;       DS - DATA SEGMENT USED BY THIS ISR
;       ES - TYMFLG SEGMENT, OR VECTOR TABLE SEGMENT
;       DI - VECTOR TABLE INDEX REGISTER
TYM_INT:
        PUSH    AX                      ;STACK UP THE REGISTERS USED
        PUSH    BX
        PUSH    DS
        PUSH    ES
        PUSH    DI
        MOV     BX,SEG TYMOFST          ;GET OUR CURRENT DATA SEGMENT
        MOV     DS,BX                   ;SETUP OUR DATA SEGMENT
        MOV     BX,TYMSEG               ;RETRIEVE TYMFLG SEGMENT AND OFFSET
        MOV     ES,BX                   ;STICK THIS SEGMENT IN ES REGISTER
        MOV     BX,TYMOFST
        MOV     AX,ES:[BX]              ;RETRIEVE THE TICKS VALUE
        DEC     AX                      ;DECREMENT TICKS VALUE
        MOV     ES:[BX],AX              ;SAVE THE NEW TICKS VALUE
        CMP     AX,0                    ;TICKS VALUE ZERO YET?
        JNZ     RTN1                    ;NO - EXIT THE ISR AS NORMAL
        XOR     AX,AX                   ;CLEAR AX
        MOV     ES,AX                   ;SETUP ES FOR VECTOR TABLE
        MOV     DI,TYMINT * 4           ;TABLE OFFSET
        MOV     AX,DUMOFST              ;RETRIEVE PREVIOUS OFFSET ADRS
        CLD                             ;SET STORE DIRECTION
        CLI                             ;DISABLE ANY INTERRUPTS
```

```
        STOSW
        MOV     AX,DUMSEG               ;RETRIEVE PREVIOUS SEG ADRS
        STOSW
        STI                             ;ENABLE THE INTERRUPTS AGAIN
RTN1:                                   ;CLEAR OFF THE STACK
        POP     DI
        POP     ES
        POP     DS
        POP     BX
        POP     AX
        IRET                            ;RETURN TO BIOS ISR
TIMER_PROC      ENDP
;
;       name:  getchr()
;       version:  1.0
;  This routine retrieves a character from the keyboard (with an echo)
;and returns this character to the caller.
        IF      S_MODEL OR D_MODEL
GETCHR_PROC     PROC    NEAR
        ELSE
GETCHR_PROC     PROC    FAR
        ENDIF

GETCHR:
        PUSH    BP                      ;STACK 'EM
        MOV     BP,SP
        MOV     AH,01H                  ;READ STATUS AND GET CHAR
        INT     16H                     ;GO FOR IT
        JNZ     CLR                     ;WE GOT IT, CLEAR OUT CHAR
        XOR     AX,AX                   ;NO CHAR, CLEAR AX
        JMP     RTN2                    ;RETURN TO CALLER
CLR:
        XOR     AH,AH                   ;ZERO 'AH' TO UPDATE KB BUF
        INT     16H                     ;GET CHAR, CLEAR KB BUFFER
RTN2:
        POP     BP
        RET                             ;ALL DONE
GETCHR_PROC     ENDP
;
;  name:  putchr(c)
;  This routine puts a character on the console
        IF      S_MODEL OR D_MODEL
PUTCHR_PROC     PROC    NEAR
        ELSE
PUTCHR_PROC     PROC    FAR
        ENDIF
PUTCHR:
        PUSH    BP                      ;STACK'EM UP
        MOV     BP,SP
        MOV     AH,14                   ;CONSOLE I/O CODE
        MOV     AL,FIRST_PARAM[BP]      ;GET THE CHAR TO DISPLAY
        MOV     BL,07H                  ;FOREGROUND COLOR, PAGE 0
```

```
        INT     10H                     ;CALL BIOS
        POP     BP
        RET
PUTCHR_PROC     ENDP
;
;       name:  display(row, col, attrib, string)
;       version:  1.0
;  This routine displays the input string at the row and column specified.
;The string is written with the attribute given.  This attribute is documented
;in the Technical Specifications manual (monochrome and printer adapter card).
;
        IF      S_MODEL OR D_MODEL
DISPLAY_PROC    PROC    NEAR
        ELSE
DISPLAY_PROC    PROC    FAR
        ENDIF

DISPLAY:
        PUSH    BP                      ;STACK 'EM UP
        MOV     BP,SP
        PUSH    SI
        PUSH    DI
        PUSH    DS
        MOV     AX,FIRST_PARAM[BP]      ;RETRIEVE THE ROW
        MOV     BX,SECOND_PARAM[BP]     ;GET THE COLUMN
        MOV     DH,AL                   ;SETUP THE REGS FOR CALL
        MOV     DL,BL
        MOV     AH,2
        MOV     BX,0
        INT     10H                     ;SET THE CURSOR LOCATION

        MOV     BX,THIRD_PARAM[BP]      ;CHARACTER ATTRIBUTE
        MOV     SI,FOURTH_PARAM[BP]     ;STRING OFFSET
      IF D_MODEL OR L_MODEL
        MOV     DS,FIFTH_PARAM[BP]      ;SEGMENT IF NEEDED
      ENDIF
        MOV     BH,0                    ;PAGE NUMBER
DLOOP:
        MOV     CX,1                    ;WRITE EACH CHARACTER ONCE
        MOV     AL,DS:[SI]              ;LOAD UP ONE CHARACTER
        INC     SI                      ;INCREMENT STRING POINTER
        CMP     AL,0                    ;DONE YET?
        JZ      RTN3                    ;YES - RETURN TO CALLER
        MOV     AH,9                    ;NO - WRITE A CHAR WITH ATTRIBUTE
        INT     10H                     ;WRITE OUT THE CHARACTER
        INC     DX                      ;UPDATE CURSOR POSITION
        MOV     AH,2
        INT     10H                     ;MOVE THE CURSOR ONE FORWARD
        JMP     DLOOP                   ;KEEP GOING UNTIL YOUR DONE
RTN3:
        POP     DS                      ;UNSTACK
        POP     DI
```

```
        POP     SI
        POP     BP
        RET                             ;RETURN TO CALLER
DISPLAY_PROC    ENDP
;
;       name:  dspln (dest_adrs, source_adrs)
;  This routine fills the graphics adapter memory with one line of data.  One
;line of data contains 640 pixels.  Upon entry the source_adrs where the data is
;currently located and the dest_adrs where the data will soon reside.
        IF      S_MODEL OR D_MODEL
DSPLN_PROC      PROC    NEAR
        ELSE
DSPLN_PROC      PROC    FAR
        ENDIF
;** parameter statements **
        COUNT   EQU     320             ;NUMBER OF BYTES IN A LINE OF DATA

DSPLN:
        PUSH    BP                      ;STACK'EM UP
        MOV     BP,SP
        PUSH    DS
        PUSH    ES
        PUSH    DI
        PUSH    SI
        MOV     AX,FIRST_PARAM[BP]      ;GET THE DEST OFFSET
        MOV     DI,AX                   ;SETUP THE INDEX
        MOV     AX,SECOND_PARAM[BP]     ;GET THE DEST SEGMENT
        MOV     ES,AX                   ;SETUP THE DESTINATION SEGMENT
        MOV     AX,THIRD_PARAM[BP]      ;GET THE SOURCE OFFSET
        MOV     SI,AX
      IF D_MODEL OR L_MODEL
        MOV     AX,FOURTH_PARAM[BP]     ;GET THE SOURCE SEGMENT
        MOV     DS,AX
      ENDIF
        MOV     CX,COUNT                ;SETUP THE LOOP COUNTER
REP     MOVSB                           ;TRANSFER THE MEMORY INTO THE GRAPHICS CARD
        MOV     AX,CX                   ;RETURN THE COUNTER VALUE
        POP     SI                      ;CLEAR OFF THE STACK
        POP     DI
        POP     ES
        POP     DS
        POP     BP
        RET                             ;NORMAL RETURN
DSPLN_PROC      ENDP
;
;       name:  clear()
;       version:  1.0
;  This routine clears the display by reseting the display mode (80 x 25).
        IF      S_MODEL OR D_MODEL
CLEAR_PROC      PROC    NEAR
        ELSE
CLEAR_PROC      PROC    FAR
```

```
        ENDIF

CLEAR:
        PUSH    BP                      ;STACK 'EM UP
        MOV     BP,SP
        MOV     AX,3                    ;80 X 25 COLOR MODE
        INT     10H                     ;CLEAR DISPLAY
        POP     BP                      ;RETURN TO CALLER
        RET
CLEAR_PROC      ENDP
;
;       name:   search (n, fcb_adrs)
;This routine searches the disk for the specified filename and returns a -1 if
;the file is not found and a 0 if the file is found
        IF      S_MODEL OR D_MODEL
SEARCH_PROC     PROC    NEAR
        ELSE
SEARCH_PROC     PROC    FAR
        ENDIF

SEARCH:
        PUSH    BP                      ;STACK 'EM UP
        MOV     BP,SP
        PUSH    DS
        MOV     AX,FIRST_PARAM[BP]      ;FUNCTION CODE (17-FIRST, 18-SECOND)
        MOV     DX,SECOND_PARAM[BP]     ;OFFSET
      IF D_MODEL OR L_MODEL
        MOV     BX,THIRD_PARAM[BP]      ;SEGMENT
        MOV     DS,BX
      ENDIF
        MOV     AH,11H
        INT     21H                     ;SEARCH DISK
        MOV     AH,AL                   ;RETURN AN INTEGER
        POP     DS
        POP     BP
        RET
SEARCH_PROC     ENDP

;
;       name:  reset()
;       version:  1.0
;  This routine disables the interrupts from the async cards.

        IF S_MODEL OR D_MODEL
RESET_PROC      PROC    NEAR
        ELSE
RESET_PROC      PROC    FAR
        ENDIF

RESET:
        IN      AL,21H                  ;GET THE CURRENT INT ENABLED
        OR      AL,08H                  ;DISABLE IRQ 3
```

```
        AND     AL,0FEH                 ;REENABLE THE TIMER INTERRUPTS
        CLI                             ;DISABLE INTERRUPTS
        OUT     21H,AL                  ;REMOVE ASYNC INT
        STI                             ;REENABLE SYSTEM
        XOR     AH,AH                   ;CLEAR HIGH AX
        RET                             ;RETURN TO CALLER
RESET_PROC      ENDP

setcom_proc     proc    near
setcom:
        push    bp
        mov     bp,sp
        mov     dx,first_param[bp]
        mov     ax,second_param[bp]
        int     14h
        pop     bp
        ret
setcom_proc     endp
;
;       name:  setread(buffer beginning adrs, buffer ending adrs)
;       version:  1.0
;  This routine initializes the interrupt vector table in the IBM lower
;memory, sets up the async. card for interrupt on character receipt,
;sets the interrupt mask on the 8259, and establishes the addresses for
;the product receipt buffer (buf[]) and product received flag
;(prodflg).  Buf[] and prodflg are externs defined in 'C'.
;
        IF S_MODEL OR D_MODEL
SETREAD_PROC    PROC    NEAR
        ELSE
SETREAD_PROC    PROC    FAR
        ENDIF

; DEFINE THE INTERRUPT VECTOR AND PORT I/O ADDRESSES
        VALU    EQU     0CH                     ;PRIMARY CARD TYPE
        PORT    EQU     3F8H                    ;CARD I/O ADDRESS

; DEFINE THE PRODUCT HEADER, TRAILER AND QUEUE FLAG ENTRIES
        SOH     EQU     01H                     ;START OF TEXT
        ETX     EQU     03H                     ;END OF TEXT
        NUMPROD EQU     3                       ;ENTRIES IN THE QUEUE
                        ;MUST BE INDENTICAL TO DEFINE.C PARAMETER

SETREAD:
        PUSH    BP                              ;SAVE THE BASE
        MOV     BP,SP
        PUSH    DI
        PUSH    ES
        MOV     AX,FIRST_PARAM[BP]              ;GET OFFSET
        MOV     BUF0,AX                         ;SAVE OFFSET
        MOV     PTR0,AX                         ;INIT POINTER
      IF D_MODEL OR L_MODEL
```

```
        MOV     AX,SECOND_PARAM[BP]             ;GET THE SEGMENT
      ELSE
        MOV     AX,DS                           ;SAME AS CURRENT SEG
      ENDIF
        MOV     BUFOSEG,AX                      ;SAVE THE SEG
      IF S_MODEL OR P_MODEL                     ;GET BUFFER ENDING LOC
        MOV     AX,SECOND_PARAM[BP]
      ELSE
        MOV     AX,THIRD_PARAM[BP]
      ENDIF
        MOV     BUFOMAX,AX                      ;SAVE THE BUF END

; INITIALIZE THE INTERRUPTS
        MOV     DX,PORT                         ;GET I/O ADRS
        ADD     DX,3                            ;CALC REGISTER ADRS
        IN      AL,DX                           ;RETRIEVE LINE STATUS
        AND     AL,07FH
        OUT     DX,AL                           ;DLAB = 0
        SUB     DX,2
        IN      AL,DX                           ;GET CURRENT INTERRUPTS
        OR      AL,01                           ;SET THE DATA AVAILABLE
        OUT     DX,AL                           ;...INTERRUPT
        ADD     DX,3                            ;MODEM CONTROL REG
        MOV     AL,08H                          ;SETUP AL
        OUT     DX,AL                           ;CLEAR MODEM CONTROL

; INITIALIZE THE SYSTEM INTERRUPT CONTROLLER (8259) AND TABLE
        XOR     AX,AX                           ;ZERO AX
        MOV     ES,AX                           ;POINT AT INT TABLE
        MOV     DI,VALU * 4                     ;INT LOCATION
        MOV     AX,OFFSET RD_INT                ;GET INTERRPT OFFSET
        CLD                                     ;STRING FORWARD DIR
        STOSW                                   ;STORE OFFSET ADRS
        MOV     AX,CS                           ;GET THE SEGMENT
        STOSW                                   ;TABLE ENTRY COMPLETE
        IN      AL,21H                          ;GET THE INT MASK
      IF VALU EQ 0CH                            ;PRIMARY PORT
        AND     AL,0EFH
      ELSE
        AND     AL,0F7H                         ;SECONDARY
      ENDIF
        CLI                                     ;DISABLE INTERRUPTS
        OUT     21H,AL                          ;SET ASYNC INT
        STI                                     ;ENABLE INTERRUPTS
        POP     ES
        POP     DI
        POP     BP
        RET                                     ;RETURN TO CALLER
;
; INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
;       AX - CONTAINS THE INPUT CHAR
```

```
;       BX - CONTAINS THE BUFFER OFFSET (BUF[])
;       CX - VARIABLE
;       DX - THE I/O PORT ADDRESS
;       DS - DATA SEGMENT FOR MEMORY ADRS USED IN THIS ROUTINE
;       ES - DATA SEGMENT OF THE BUFFER (BUF[])
RD_INT:
        PUSH    ES                      ;STACK REGISTERS USED
        PUSH    DS
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     DX,PORT                 ;SETUP PORT I/O ADRS
        IN      AL,DX                   ;READ CHAR
        ADD     DX,5                    ;LINE STATUS ADRS
        PUSH    AX                      ;SAVE CHAR
        IN      AL,DX                   ;READ LINE STATUS
        AND     AL,017H                 ;ISOLATE ERROR BITS
        POP     AX                      ;RESTORE CHAR
        JZ      SKIP1                   ;ERROR?
        MOV     AL,'?'                  ;REPLACE DATA WITH '?'
SKIP1:                                  ;NO ERROR, CONTINUE ON
        MOV     BX,SEG BUF0             ;GET THE DATA SEG
        MOV     DS,BX                   ;SETUP DS ADRSING
        MOV     BX,BUF0SEG              ;RETRIEVE BUF SEG
        MOV     ES,BX                   ;SETUP ES FOR BUF
        MOV     BX,PTR0                 ;GET CURRENT OFFSET
        MOV     ES:[BX],AL              ;PUT THE CHAR IN BUF
        INC     BX                      ;INCREMENT POINTER
        MOV     CX,BUF0MAX              ;GET THE BUF END OFFSET
        CMP     BX,CX                   ;AT THE BUFFER END?
        JB      SKIP2                   ;NO - CONTINUE ON
        MOV     BX,BUF0                 ;YES - RESET PTR
SKIP2:
        MOV     PTR0,BX                 ;UPDATE IN MEMORY
        CMP     AL,SOH                  ;BEGINNING OF PROD?
        JZ      BEGIN
        CMP     AL,ETX                  ;END OF PROD?
        JZ      FINISHED
RETURN:
        CLI                             ;DISABLE INTERRUPTS
        MOV     AL,20H                  ;CLEAR INTERRUPT
        OUT     20H,AL
        POP     DX
        POP     CX                      ;CLEAR STACK
        POP     BX
        POP     AX
        POP     DS
        POP     ES
        IRET                            ;RESUME
BEGIN:                                  ;START OF A NEW PROD
        MOV     CX,BUF0                 ;GET THE BUF BEGINNING
```

```
        SUB     BX,CX                           ;CALC ARRAY ELEMENT + 1
        MOV     BEGINO,BX                       ;SAVE STARTING ELEMENT
        JMP     RETURN
FINISHED:                                       ;END OF PRODUCT
        MOV     AX,BEGINO                       ;GET STARTING ELEMENT
        XOR     CX,CX                           ;CLEAR 'CX'
        MOV     BEGINO,CX                       ;CLEAR BEGINNING INDICE
        CMP     AX,00H                          ;VALID PROD START?
        JZ      RETURN                          ;IF NOT, FLUSH TRASH
        MOV     BX,SEG PRODFLG                  ;LOAD PRODFLG SEG
        MOV     DS,BX                           ;DS IS NOW PRODFLG SEG
        MOV     BX,OFFSET PRODFLG               ;ACTUAL PRODFLG ADRS
        MOV     DX, [BX]                        ;GET THE CURRENT ENTRY VAL
        CMP     DX, 00H                         ;ENTRY EMPTY?
        JZ      FLGMAIN                         ;YES - FLAG PROD FOR MAIN()
        MOV     CX, NUMPROD - 1                 ;NO - SETUP COUNTER FOR # OF QUE ENTRIES
NEXT:
        INC     BX                              ;TRY ANOTHER
        INC     BX
        MOV     DX, [BX]                        ;GET THE NEXT ENTRY
        CMP     DX, 00H                         ;ENTRY EMPTY?
        LOOPNZ  NEXT                            ;TRY ANOTHER
FLGMAIN:
        MOV     [BX], AX                        ;SET THE PRODFLG WITH LOC...
                                                ;OF THE *BUFFER+1
        JMP     RETURN
SETREAD_PROC    ENDP

;
;       name:  tx_cont (sw_int, &tx_air, &silence)
;  This routine setups up a software interrupt to service the Vynet continuous
;speech device driver.  Upon entry sw_int contains the software interrupt number,
;&tx_air contains the starting address of the buffer control structure, and
;&silence contains the address of a silence buffer for use when all the transmit
;buffers are empty.
;
        IF S_MODEL OR D_MODEL
TX_CONT_PROC    PROC    NEAR
        ELSE
TX_CONT_PROC    PROC    FAR
        ENDIF
;parameters:  these values must correspond to define.h values
;       define the control bit settings
        EMPTY   EQU     00H                     ;memory buffer is empty
        FULL    EQU     01H                     ;memory buffer contains voice data
        IN_USE  EQU     02H                     ;memory buffer is currently being used
        SKIP    EQU     04H                     ;memory buffer is a continuation of a previous buffer
        ABORT   EQU     08H                     ;stop the speech driver
;
        TX_BNUM EQU     8                       ;number of control buffers

TX_CONT:
```

```
        PUSH    BP                              ;STACK'EM UP
        MOV     BP,SP
        PUSH    DI
        PUSH    ES
;
;  SAVE THE REQUIRED DATA ADDRESSES
        MOV     AX,SECOND_PARAM[BP]             ;GET THE CTRL STRUCT OFFSET
        MOV     TXOFST,AX                       ;SAVE THIS OFFSET
      IF S_MODEL OR P_MODEL
        HLT                                     ;CODE NOT WRITTEN, SO A MONUMENTAL BUG THAT NOBODY CAN
                                                ;MISS IS PUT IN ITS PLACE
      ELSE
        MOV     AX,THIRD_PARAM[BP]              ;GET THE CTRL STRUCT SEG
        MOV     TXSEG,AX
        MOV     AX,FIFTH_PARAM[BP]              ;GET THE SILENCE SEG
        MOV     DX,16                           ;CONVERT TO A PHYSICAL ADRS
        MUL     DX
        ADD     AX,FOURTH_PARAM[BP]             ;ADD IN THE OFFSET
        ADC     DX,0                            ;ADJUST FOR CARRY
        MOV     SILOFST,AX                      ;SAVE THE PHYSICAL OFFSET
        XOR     AX,AX                           ;CLEAR AX
        MOV     AL,DL                           ;GET BITS 16 - 23
        MOV     SILSEG,AX                       ;SAVE THE OFFSET
      ENDIF
;
;  INITIALIZE THE INTERRUPT TABLE
        MOV     AX,FIRST_PARAM[BP]              ;PUT THE SOFTWARE INT NUMBER IN AX
        SHL     AX,1                            ;MULTIPLY AX BY 4
        SHL     AX,1
        MOV     DI,AX                           ;POINT AT THE INT LOC
        XOR     AX,AX
        MOV     ES,AX                           ;TABLE ADRS COMPLETE
        MOV     AX,OFFSET TX_INT                ;GET THE ISR OFFSET
        CLD                                     ;STRING FORWARD DIR
        STOSW                                   ;STORE OFFSET
        MOV     AX,CS                           ;GET THE CODE SEG
        STOSW                                   ;TABLE ENTRY COMPLETE
        POP     ES                              ;THATS ALL - UNSTACK'EM
        POP     DI
        POP     BP
        RET
;
; INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
;       AX - VARIABLE, AND THE RETURN CODE
;       BX - BUFFER CONTROL STRUCTURE OFFSET
;       CX - COUNTER
;       DX - IN_USE BUFFER OFFSET
;       DS - DATA SEGMENT FOR TX ASSEMBLER VARIABLES
;       ES - BUFFER CONTROL STRUCTURE SEGMENT
TX_INT:
        STI                                     ;ENABLE OTHER INTERRUPTS
```

```
        PUSH    DS                      ;PRESERVE THE CURRENT STATE
        PUSH    CX
        PUSH    DX
        PUSH    BX
        PUSH    ES
;
;  ESTABLISH THIS ROUTINES ADDRESSING REGISTERS
        MOV     AX,SEG TXOFST           ;GET OUR DATA SEGMENT
        MOV     DS,AX                   ;ESTABLISH OUR SEG FOR USE
        MOV     AX,TXSEG                ;GET THE CONTROL STRUCT SEG
        MOV     ES,AX                   ;ES NOW CONTAINS THE SEG OF THE BUFFER CONTROL STRUCTURE
        MOV     BX,TXOFST               ;GET THE CTRL STRUCT OFFSET
        MOV     CX,TX_BNUM              ;COUNTER IS NOW EQUAL TO THE NUMBER OF BUFFERS
;
;  FIND THE CURRENT (LAST) BUFFER USED
FIND_USE:
        MOV     AL,ES:[BX]              ;GET THE CTRL BITS
        TEST    AL,IN_USE               ;TEST FOR THE IN_USE BIT
        JNZ     FOUND_USE
        TEST    AL,ABORT                ;NO IN_USE, ABORT CONDITION?
        JNZ     STOP
        ADD     BX,5                    ;GET THE NEXT RECORD OFFSET
        LOOPNZ  FIND_USE                ;KEEP LOOPING
        JNZ     STOP                    ;REAL PROBLEMS IF WE DROP THRU THIS LOOP
;
;  FIND THE NEXT BUFFER TO BROADCAST
FOUND_USE:
        MOV     DX,BX                   ;SAVE THE IN_USE FLAG OFFSET
        ADD     BX,5                    ;GET THE NEXT RECORD OFFSET
        DEC     CX                      ;UPDATE COUNTER TO SKIP IN_USE FLAG
        MOV     AH,CL                   ;SAVE THE CURRENT COUNT IN AH
        CMP     CX,0                    ;ARE WE AT THE END?
        JNZ     FIND_FL1                ;NO - CONTINUE THE SEARCH
        MOV     BX,TXOFST               ;YES - RESET OFFSET TO THE BEGINNING
        MOV     CX,TX_BNUM              ;RESET THE COUNTER
        JMP     FIND_FL2                ;ONLY NEED TO MAKE ONE PASS THRU THE STRUCTURE
FIND_FL1:                               ;START THE SEARCH FOR A FULL BUFFER
        MOV     AL,ES:[BX]              ;GET THE CTRL BITS
        TEST    AL,FULL                 ;IS THIS BUFFER READY?
        JNZ     RTN6                    ;YES - PROVIDE INFO TO CALLER
        CMP     AL,EMPTY                ;EMPTY BUFFER?
        JZ      RTN7                    ;YES - ALLOW TX_BUF() TO CATCH UP WITH A PAUSE
        TEST    AL,SKIP                 ;BUFFER A CONTINUATION OF PREV BUFFER?
        JZ      SKIP4                   ;NO - CONTINUE ON
        MOV     AL,EMPTY                ;ZERO AL
        MOV     ES:[BX],AL              ;CLEAR THE SKIP FLAG
SKIP4:
        ADD     BX,5                    ;ADJUST OFFSET
        LOOPNZ  FIND_FL1                ;LOOP UNTIL WE FIND A BUFFER
        MOV     BX,TXOFST               ;RECYCLE POINTER TO THE BEGINNING
        MOV     CX,TX_BNUM              ;RESET THE COUNTER
        SUB     CL,AH                   ;NO USE CHECKING WHAT WE HAVE ALREADY CHECKED
```

```
FIND_FL2:
        MOV     AL,ES:[BX]                      ;GET THE CTRL BITS
        TEST    AL,FULL                         ;IS THIS BUFFER READY?
        JNZ     RTN6                            ;YES - RETURN INFO TO CALLER
        CMP     AL,EMPTY                        ;ARE WE AHEAD OF TX_BUF()?
        JZ      RTN7                            ;YES - ALLOW TX_BUF() TO CATCH UP
        TEST    AL,SKIP                         ;IS THIS A BUFFER TO SKIP?
        JZ      SKIP5                           ;NO - NO REASON TO RESET BITS
        MOV     AL,EMPTY                        ;ZERO AL
        MOV     ES:[BX],AL                      ;RESET BITS
SKIP5:
        ADD     BX,5                            ;ADJUST OFFSET
        LOOPNZ  FIND_FL2                        ;CHECK THE REMAINING BUFFERS
;
; NO BUFFERS AVAILABLE, PAUSE FOR TX_BUF() TO CATCH UP
RTN7:
        POP     ES                              ;REMOVE ES,BX FROM THE STACK
        POP     BX
        MOV     AX,SILSEG
        MOV     ES,AX                           ;ESTABLISH THE SILENCE SEGMENT
        MOV     AX,SILOFST
        MOV     BX,AX                           ;ESTABLISH THE SILENCE BUFFER OFFSET
        XOR     AX,AX                           ;RETURN THE CONTINUE CODE
        JMP     RTN8                            ;RETURN INFO TO CALLER
;
; STOP THE SPEECH
STOP:
        MOV     AX,-1                           ;RETURN THE STOP CODE OF -1 IN AX
        POP     ES                              ;RESTORE THE PREVIOUS ES,BX REGS
        POP     BX
        JMP     RTN8                            ;RETURN INTO TO CALLER
;
; GIVE THE NEW BUFFER ADRSES TO THE CALLER
RTN6:
        MOV     AL,IN_USE                       ;GET THE IN_USE FLAG BITS
        MOV     ES:[BX],AL                      ;SET THE NEW BUFFER AS IN_USE
        MOV     CX,ES:[BX+1]                    ;GET THE OFFSET
        MOV     AX,ES:[BX+3]                    ;GET THE SEG ADRS
        MOV     BX,DX                           ;GET THE LAST IN_USE BUFFER
        MOV     DX,AX                           ;PUT THE SEG ADRS IN DX
        MOV     AL,EMPTY                        ;ZERO AL
        MOV     ES:[BX],AL                      ;CLEAR THE LAST IN_USE FLAG
        POP     ES                              ;CLEAR ES,BX OFF THE STACK
        POP     BX
        MOV     AX,16                           ;CONVERT ADRS TO A PHYSICAL ADRS
        MUL     DX
        ADD     AX,CX                           ;ADD IN THE OFFSET
        ADC     DX,0                            ;ACCOUNT FOR THE CARRY
        MOV     BX,AX                           ;ESTABLISH THE REGS FOR CALLER
        MOV     ES,DX
        XOR     AX,AX                           ;RETURN THE CONTINUE CODE
RTN8:
```

```
        POP     DX                              ;CLEAR OFF THE STACK
        POP     CX
        POP     DS
        IRET                                    ;RETURN INFO TO THE CALLER

TX_CONT_PROC    ENDP

;       name:  tx_chk (&tx_air)
;  This routine disables the interrupts to check for empty transmit buffers
;which should be filled with data.  If an empty buffer is found the buffer number
;is returned to the caller.  If no empty buffers exist a -1 is returned.
;
        IF S_MODEL OR D_MODEL
TX_CHK_PROC     PROC    NEAR
        ELSE
TX_CHK_PROC     PROC    FAR
        ENDIF

;DEFINE THE NUMBER OF BUFFERS AND THE RETURN CODE
;       TX_BNUM EQU     10              ;DEFINED PREVIOUSLY IN TX_CONT
;       IN_USE  EQU     2               ;DITTO...
;       EMPTY   EQU     0
        FALSE   EQU     -1              ;NO AVAILABLE BUFFERS RETURN CODE

TX_CHK:
        PUSH    BP                              ;STACK'EM UP
        MOV     BP,SP
        PUSH    ES
        MOV     BX,FIRST_PARAM[BP]              ;CTRL STRUCT OFFSET ADRS
        MOV     DX,BX                           ;SAVE AN ADDITIONAL COPY OF THE OFFSET
      IF D_MODEL OR L_MODEL
        MOV     AX,SECOND_PARAM[BP]             ;SEGMENT ADRS
      ELSE
        MOV     AX,DS                           ;USE THE CURRENT SEGMENT
      ENDIF
        MOV     ES,AX                           ;PUT SEGMENT IN ES
        MOV     CX,TX_BNUM                      ;SETUP THE COUNTER
        CLI                                     ;DISABLE THE INTERRUPTS
;
;  FIND THE IN_USE FLAG
PASS1:
        MOV     AL,ES:[BX]                      ;GET THE CTRL BITS
        TEST    AL,IN_USE                       ;IN_USE FLAG?
        JNZ     CONT1                           ;FOUND, CHECK FOR AN EMPTY
        ADD     BX,5                            ;UPDATE POINTER, CLEAR ZF
        LOOPNZ  PASS1                           ;NOT FOUND, CONTINUE ON
        MOV     AX,FALSE                        ;PROBLEMS IF WE DROP THRU
        JMP     RTN11                           ;RETURN A FALSE ANSWER
CONT1:
        DEC     CX                              ;ADJUST COUNTER
        MOV     AH,CL                           ;SAVE THE CURRENT COUNTER
        CMP     CX,0                            ;CYCLE TO THE BEGINNING?
```

```
        JZ      CONT2                           ;YES
        ADD     BX,5                            ;UPDATE THE POINTER
;
; FIND AN EMPTY BUFFER IF IT EXISTS
PASS2:
        MOV     AL,ES:[BX]                      ;GET THE CTRL BITS
        CMP     AL,EMPTY                        ;EMPTY BUFFER?
        JZ      RTN10                           ;YES - RETURN TO CALLER
        ADD     BX,5                            ;UPDATE POINTER
        LOOPNZ  PASS2                           ;NO - CONTINUE CHECKING
CONT2:
        MOV     CX,TX_BNUM                      ;SETUP THE COUNTER TO CYCLE
        SUB     CL,AH                           ;NO USE CHECK WHAT HAS ALREADY BEEN DONE
        MOV     BX,DX                           ;RESET THE POINTER TO THE BUFFER BEGIN
PASS3:                                          ;FINAL PASS THRU THE CTRL BITS
        MOV     AL,ES:[BX]                      ;SAME LOGIC AS PASS2
        CMP     AL,EMPTY
        JZ      RTN9
        ADD     BX,5
        LOOPNZ  PASS3
        MOV     AX,FALSE                        ;RETURN N/A TO CALLER
        JMP     RTN11
RTN9:
        ADD     CL,AH                           ;GET THE # OF RECORDS LEFT IN THE STRUCT
RTN10:                                          ;RETURN THE BUFFER NUMBER TO CALLER
        MOV     AX,TX_BNUM
        SUB     AX,CX                           ;GET THE RECORD NUMBER
RTN11:
        STI                                     ;REENABLE THE INT
        POP     ES
        POP     BP
        RET

TX_CHK_PROC     ENDP

;
;       name:  satdtgl (start_buf_adrs, end_buf_adrs)
;  This routine sets up the A/D converter for digitizing a demodulated signal,
;initializes the interrupt controller and table, and contains the interrupt routine
;for receiving the digitized data into RAM.  Upon the entry, this routine must
;receive the starting buffer address and the ending buffer address.  A limit is
;placed on the buffer of 1 segment, or 64K.  The buffer is circular and continuous,
;so data is constantly being placed in the buffer.
;
        IF S_MODEL OR D_MODEL
SATDGTL_PROC    PROC    NEAR
        ELSE
SATDGTL_PROC    PROC    FAR
        ENDIF

;DEFIN THE INTERRUPT VECTOR AND BASE ADDRESS
        IRQ     EQU     3                       ;USE THE PRIMARY ASYNC VECTOR
```

```
        BASE    EQU     2f8H                    ;DT2814 JUMPERED TO THIS ADRS
        FULL_LN EQU     1500                    ;NUMBER OF SAMPLES IN A LINE OF DATA

SATDGTL:
        PUSH    BP                              ;STACK'EM UP
        MOV     BP,SP
        PUSH    DI
        PUSH    ES
        MOV     AX,FIRST_PARAM[BP]              ;GET THE STARTING OFFSET
        MOV     SBUF_OFST, AX                   ;SAVE IN MEMORY FOR THE INTERRPT ROUTINE
        MOV     BUF_PTR, AX                     ;INIT BUFFER POINTER
      IF D_MODEL OR L_MODEL                     ;DETERMINE ADRSING LENGTH
        MOV     AX,SECOND_PARAM[BP]             ;GRAB THE SEGMENT
        MOV     BX,THIRD_PARAM[BP]              ;GRAB THE ENDING OFFSET
      ELSE
        MOV     AX,DS                           ;ADRSING SEG IS THE CURRENT ONE
        MOV     BX,SECOND_PARAM[BP]             ;GET THE ENDING OFFSET
      ENDIF
        MOV     BUF_SEG,AX                      ;SAVE THE SEG FOR ISR
        MOV     EBUF_OFST,BX                    ;SAVE ENDING OFST FOR ISR
;
; INITIALIZE THE ISR VARIABLES FOR DATA STORAGE
        MOV     STORFLG,01H                     ;SET THE STORE FLAG TO ON
        MOV     PIXELS,FULL_LN                  ;INIT NUMBER OF SAMPLES TO STORE

;
; INITIALIZE THE SYSTEM INTERRUPT CONTROLLER AND TABLE
        XOR     AX,AX                           ;ZERO AX
        MOV     ES,AX                           ;POINT AT THE INT TABLE
        MOV     DI,(IRQ + 8) * 4                ;INT LOCATION
        MOV     AX,OFFSET ATOD_INT              ;GET THE ROUTINE OFFSET
        CLD                                     ;STRING FORWARD DIR
        STOSW                                   ;PLACE OFFSET IN TABLE
        MOV     AX,CS                           ;GET CODE SEGMENT
        STOSW                                   ;PLACE SEGMENT IN TABLE
        IN      AL,21H                          ;GET THE CURRENT INTERRUPTS
        MOV     BL,01H                          ;SET BIT ZERO ON
        MOV     CL,IRQ                          ;LOAD COUNTER
        SAL     BL,CL                           ;SET THE IRQ BIT
        NOT     BL                              ;FLIP BITS SO IRQ BIT IS 0
        AND     AL,BL                           ;ENABLE THE IRQ BIT
;       OR      AL,01H                          ;DISABLE THE TIMER INTERRUPTS
        CLI                                     ;DISABLE ANY INTERRUPTS
        OUT     21H,AL                          ;LET IRQ 3 THROUGH
        STI                                     ;ENABLE ALL INTERRUPTS BACK
        POP     ES                              ;CLEAR'EM OUT
        POP     DI
        POP     BP
        RET                                     ;RETURN TO CALLER
;
; A/D CONVERSION INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
```

```
;       AX - CONTAINS THE DIGITAL DATA OUTPUT
;       BX - STATUS BITS AND THE DATA BUFFER OFFSET (PTR)
;       CX - VARIABLE
;       DX - THE I/O PORT ADRS
;       DS - DATA SEGMENT OF THE STORAGE VARIABLES USED HERE
;       ES - DATA SEGMENT OF THE DATA BUFFER
ATOD_INT:
        PUSH    ES                              ;STACK'EM
        PUSH    DS
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
;
; RETRIEVE DATA, CHECK STATUS AND DETERMINE ACTIONS TO TAKE WITH THE DATA
        MOV     DX,BASE                         ;STATUS INPUT ADRS
        IN      AL,DX                           ;GET THE STATUS BYTE
        MOV     BX,AX                           ;MOVE STATUS BYTE OVER TO BX
        MOV     DX,BASE + 1                     ;GET THE DATA REG ADRS
        IN      AL,DX                           ;READ IN FIRST DATA BYTE
        MOV     CX,AX                           ;SAVE DATA IN CX
        IN      AL,DX                           ;READ IN THE SECOND BYTE AND TOSS
        MOV     AX,CX                           ;PUT THE GOOD BYTE BACK IN AX
        MOV     CX,SEG BUF_PTR                  ;GET OUR VARIABLE DATA SEG
        MOV     DS,CX                           ;PLACE SEG IN DS FOR USE
        CMP     STORFLG,01H                     ;ARE WE IN STORAGE MODE?
        JE      STORE                           ;YES - STORE THE DATA
        DEC     PIXELS                          ;NO - KEEP TRACK OF OUR LOC
        JNZ     RTN5                            ;RETURN TO PREV TASK IF MORE DATA TO TOSS
        MOV     STORFLG,01H                     ;DONE TOSSING DATA, FLIP STORE FLAG
        MOV     PIXELS,FULL_LN                  ;SETUP THE NUMBER OF SAMPLES TO STORE
        JMP     RTN5                            ;READY TO STORE DATA NOW
;
; STORE THE DATA IN THE BUFFER
STORE:
        AND     BL,40H                          ;CHECK THE STATUS ERROR BIT
        JZ      SKIP3                           ;DID AN ERROR OCCUR?
        MOV     AL, 0FFH                        ;YES - LET BAD DATA BE BLACK
SKIP3:
        MOV     BX,BUF_SEG                      ;GET THE DATA BUFFER SEG
        MOV     ES,BX                           ;SETUP ES FOR THE DATA BUFFER
        MOV     BX,BUF_PTR                      ;GET THE CURRENT BUFFER POINTER
        MOV     ES:[BX],AL                      ;STORE THE DATA IN THE BUFFER
        CALL    CHECK                           ;UPDATE BUFFER POINTER
;
; CHECK THE AMOUNT OF DATA IN THIS LINE
        DEC     PIXELS                          ;UPDATE THE COUNTER
        JNZ     RTN4                            ;RETURN IF NOT DONE YET
        MOV     STORFLG,00H                     ;WE ARE DONE, FLIP STORE FLAG
        MOV     PIXELS,FULL_LN * 3              ;SKIP THE NEXT THREE LINES
        INC     LINENUM                         ;UPDATE LINE NUMBER
        MOV     AX,LINENUM                      ;BRING THE LINE NUMBER INTO A REG
```

```
        MOV     ES:[BX],AL                      ;STORE LINE NUM IN DATA BUFFER
        CALL    CHECK                           ;CHECK DATA POINTER
        MOV     CL,8                            ;SETUP COUNTER FOR ROTATING
        ROR     AX,CL                           ;PUT HIGH BYTE IN LOW REG
        MOV     ES:[BX],AL                      ;FINISH STORING LINE NUMBER
        CALL    CHECK                           ;UPDATE POINTER
RTN4:                                           ;RETURN TO THE PREV TASK
        MOV     BUF_PTR,BX                      ;SAVE POINTER IN MEMORY
;
; RETURN THE PC TO ITS PREVIOUS TASK
RTN5:
        MOV     AL,20H                          ;LET THE 8259 KNOW WE'RE DONE
        CLI                                     ;DISABLE INTERRUPTS TO BE SURE
        OUT     20H,AL                          ;CLEAR THE INTERRUPT
        POP     DX                              ;CLEAR OFF THE STACK
        POP     CX
        POP     BX
        POP     AX
        POP     DS
        POP     ES
        IRET
;
; SUBROUTINE TO UPDATE AND CHECK THE DATA BUFFER POINTER
;       UPON ENTRY:     BX - CONTAINS THE POINTER
;                       DS - CONTAINS THE ISR DATA SEGMENT
;       UPON EXIT:      BX - CONTAINS THE UPDATED POINTER
;                       NO OTHER REGISTERS ARE CHANGED
CHECK:
        INC     BX                              ;UPDATE THE POINTER
        CMP     BX,EBUF_OFST                    ;COMPARE WITH THE END OF BUFFER
        JB      CHKRTN                          ;RETURN IF NOT PAST THE END
        MOV     BX,SBUF_OFST                    ;OTHERWISE RESET TO THE BEGINNING
CHKRTN:
        RET                                     ;RETURN TO CALLER

SATDGTL_PROC    ENDP
;
;*************************************************************************
; end the appropriate segment
        IF      S_MODEL
PROG    ENDS
        ENDIF

        IF      P_MODEL
_CODE   ENDS
        ENDIF

        IF      D_MODEL
CODE    ENDS
        ENDIF

        IF      L_MODEL
```

```
_PROG   ENDS
        ENDIF

        END                     ;(END OF FILE)
```

```
;       date:  May 1987
;       version:  1.0
;       by:  Glen W. Sampson, NWS WRH SSD
;
;Purpose:
;  These routines contains a BIOS function call not normally available
;in C, and a device driver for the asynchronous communication adapters.
;
;Exits:
;  No exits exist from these routines.
;
;Created by:
;       (Version 1.0 IBM Macro Assembler)
;       C>masm func, func, nul, nul
;
;************************************************************************
; setup the proper memory model
S_MODEL         EQU     0
P_MODEL         EQU     0
D_MODEL         EQU     1
L_MODEL         EQU     0
;
;************************************************************************
; setup the stack offsets
        IF      L_MODEL OR P_MODEL              ;FAR CALLS
FIRST_PARAM     EQU     6
SECOND_PARAM    EQU     8
THIRD_PARAM     EQU     10
FOURTH_PARAM    EQU     12
FIFTH_PARAM     EQU     14
SIXTH_PARAM     EQU     16
        ELSE
FIRST_PARAM     EQU     4                       ;NEAR CALLS
SECOND_PARAM    EQU     6
THIRD_PARAM     EQU     8
FOURTH_PARAM    EQU     10
FIFTH_PARAM     EQU     12
SIXTH_PARAM     EQU     14
        ENDIF
;
; ***********************************************************************
; declare external functions (if appropriate)
        IF P_MODEL OR L_MODEL
;       EXTRN   FUNCTION:FAR
        ENDIF
;
; ***********************************************************************
; establish the DATA SEGMENT
DGROUP  GROUP   DATA
DATA    SEGMENT WORD    PUBLIC  'DATA'
        ASSUME  DS:DGROUP
;
```

```
; Function variables
;
;       asynchronous function variables
;               receive
        EXTRN           OUT_PTR:WORD                    ;output buffer offset
        BUFSEG          DW      00H                     ;buffer segment
        IN_PTR          DW      00H                     ;input buffer offset
        IN_START        DW      00H                     ;start of the buffer offset
        IN_MAX          DW      00H                     ;maximum buffer offset
        PORT            DW      00H                     ;i/o port adrs

DATA    ENDS                                            ;end data segment
;
;*************************************************************************
; establish the GROUP and SEGMENT definitions
        IF      S_MODEL
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE    PUBLIC  'PROG'
        ASSUME  CS:PGROUP
        ENDIF

        IF      P_MODEL
PGROUP  GROUP   CODE
_CODE   SEGMENT BYTE    PUBLIC  '_CODE'
        ASSUME  CS:PGROUP
        ENDIF

        IF      D_MODEL
CGROUP  GROUP   CODE
CODE    SEGMENT BYTE    PUBLIC  'CODE'
        ASSUME  CS:CGROUP
        ENDIF

        IF      L_MODEL
CGROUP  GROUP   _PROG
_PROG   SEGMENT BYTE    PUBLIC  '_PROG'
        ASSUME  CS:CGROUP
        ENDIF

;
;*************************************************************************
; put the hooks in for 'C' access
        PUBLIC  SETAOS
                PUBLIC IN_PTR
                PUBLIC IN_MAX
                PUBLIC IN_START
        PUBLIC  GETCUR
;
;
;*************************************************************************
;
;                       f u n c t i o n s
```

```
;
;*************************************************************************
;
;       name:  setaos(port adrs, buffer size, irq #, buffer adrs)
;  This routine initializes the interrupt vector table in the IBM lower
;memory, sets up the async. card for interrupt on character receipt,
;sets the interrupt mask on the 8259, and establishes the addresses for
;the product receipt buffer (buf[]).
;
        IF S_MODEL OR D_MODEL
SETAOS_PROC     PROC    NEAR
        ELSE
SETAOS_PROC     PROC    FAR
        ENDIF
        base    equ     3f8h
SETAOS:
        PUSH    BP                              ;SAVE THE BASE
        MOV     BP,SP
        PUSH    DI
        PUSH    DS
        PUSH    ES
        MOV     AX,FIRST_PARAM[BP]              ;GET PORT
        MOV     PORT,AX                         ;SAVE THE PORT ADRS
        MOV     AX,FOURTH_PARAM[BP]             ;GET THE BUFFER OFFSET
        MOV     IN_MAX,AX                       ;PUT INIT VALUE IN ENDING POINT
        MOV     IN_PTR,AX                       ;INIT C POINTERS
        MOV     OUT_PTR,AX
        MOV     IN_START,AX                     ;INIT STARTING POINT
      IF D_MODEL OR L_MODEL
        MOV     AX,FIFTH_PARAM[BP]              ;GET THE SEGMENT
      ELSE
        MOV     AX,DS                           ;SAME AS CURRENT SEG
      ENDIF
        MOV     BUFSEG,AX                       ;SAVE THE SEG
        MOV     AX,SECOND_PARAM[BP]             ;GET THE BUFFER SIZE
        ADD     IN_MAX,AX                       ;GET THE ENDING POINT

; INITIALIZE THE INTERRUPTS
        MOV     DX,PORT                         ;GET I/O ADRS
        ADD     DX,3                            ;CALC REGISTER ADRS
        IN      AL,DX                           ;RETRIEVE LINE STATUS
        AND     AL,07FH
        OUT     DX,AL                           ;DLAB = 0
        SUB     DX,2
        IN      AL,DX                           ;GET CURRENT INTERRUPTS
        OR      AL,01                           ;SET THE DATA AVAILABLE
        OUT     DX,AL                           ;...INTERRUPT
        ADD     DX,3                            ;MODEM CONTROL REG
        MOV     AL,0BH                          ;SETUP AL
        OUT     DX,AL                           ;CLEAR MODEM CONTROL

; INITIALIZE THE SYSTEM INTERRUPT CONTROLLER (8259) AND TABLE
```

```
        XOR     AX,AX                   ;ZERO AX
        MOV     ES,AX                   ;POINT AT INT TABLE
        MOV     AX,THIRD_PARAM[BP]      ;GET THE IRQ
        ADD     AX,8                    ;GET THE TYPE CODE
        SHL     AX,1                    ;MULTIPLY BY 4 FOR TABLE LOC
        SHL     AX,1
        MOV     DI,AX                   ;INT LOCATION
        MOV     AX,OFFSET AOS_INT       ;GET INTERRPT OFFSET
        CLD                             ;STRING FORWARD DIR
        STOSW                           ;STORE OFFSET ADRS
        MOV     AX,CS                   ;GET THE SEGMENT
        STOSW                           ;TABLE ENTRY COMPLETE
        IN      AL,21H                  ;GET THE INT MASK
        MOV     BL,01H                  ;SET BIT ONE
        MOV     CL,THIRD_PARAM[BP]      ;SET COUNTER WITH IRQ VALUE
        SAL     BL,CL                   ;SET THE IRQ BIT
        NOT     BL                      ;FLIP BITS SO IRQ IS 0
        AND     AL,BL                   ;ENABLE THE IRQ BIT
        CLI                             ;DISABLE INTERRUPTS
        OUT     21H,AL                  ;SET ASYNC INT
        STI                             ;ENABLE INTERRUPTS
        POP     ES
        POP     DS
        POP     DI
        POP     BP
        RET                             ;RETURN TO CALLER
;
; INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
;       AX - CONTAINS THE INPUT CHAR
;       BX - CONTAINS THE BUFFER OFFSET (BUF[])
;       CX - VARIABLE
;       DX - THE I/O PORT ADDRESS
;       DS - DATA SEGMENT FOR MEMORY ADRS USED IN THIS ROUTINE
;       ES - DATA SEGMENT OF THE BUFFER (BUF[])
AOS_INT:
        PUSH    ES                      ;STACK REGISTERS USED
        PUSH    DS
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     BX,SEG BUFSEG           ;ESTABLISH THE PROPER DATA SEG
        MOV     DS,BX
        MOV     DX,PORT                 ;SETUP PORT I/O ADRS
        IN      AL,DX                   ;READ CHAR
        ADD     DX,5                    ;LINE STATUS ADRS
        PUSH    AX                      ;SAVE CHAR
        IN      AL,DX                   ;READ LINE STATUS
        AND     AL,017H                 ;ISOLATE ERROR BITS
        POP     AX                      ;RESTORE CHAR
        JZ      SKIP1                   ;ERROR?
```

```
        MOV     AL,'?'                          ;REPLACE DATA WITH '?'
SKIP1:                                          ;NO ERROR, CONTINUE ON
        MOV     BX,BUFSEG                       ;RETRIEVE BUF SEG
        MOV     ES,BX                           ;SETUP ES FOR BUF
        MOV     BX,IN_PTR                       ;GET CURRENT OFFSET
        MOV     ES:[BX],AL                      ;PUT THE CHAR IN BUF
        INC     BX                              ;INCREMENT POINTER
        MOV     CX,IN_MAX                       ;GET THE BUF END OFFSET
        CMP     BX,CX                           ;AT THE BUFFER END?
        JB      SKIP2                           ;NO - CONTINUE ON
        MOV     BX,IN_START                     ;YES - RESET PTR
SKIP2:
        MOV     IN_PTR,BX                       ;UPDATE IN MEMORY
        CLI                                     ;DISABLE INTERRUPTS
        MOV     AL,20H                          ;CLEAR INTERRUPT
        OUT     20H,AL
        POP     DX
        POP     CX                              ;CLEAR STACK
        POP     BX
        POP     AX
        POP     DS
        POP     ES
        IRET                                    ;RESUME

SETAOS_PROC     ENDP

;       name:  get_cursr
;  This routine retrieves the current location of the cursor on the screen.

        IF S_MODEL OR D_MODEL
GETCUR_PROC     PROC    NEAR
        ELSE
GETCUR_PROC     PROC    FAR
        ENDIF

GETCUR:
        PUSH    BP                              ;STACK'EM UP
        MOV     BP,SP
        MOV     AH,3                            ;READ CURSOR CODE
        MOV     BH,0                            ;SET TO PAGE 0
        INT     10H                             ;CALL VIDEO I/O ROM
        MOV     AX,DX                           ;MOV RESULT FOR RETURN
        POP     BP
        RET
GETCUR_PROC     ENDP
;
;*******************************************************************************
; end the appropriate segment
        IF      S_MODEL
PROG    ENDS
        ENDIF
```

```
        IF      P_MODEL
_CODE   ENDS
        ENDIF

        IF      D_MODEL
CODE    ENDS
        ENDIF

        IF      L_MODEL
_PROG   ENDS
        ENDIF
        END                                     ;END OF FILE
```

5. Data Expansion (SATEXPND)

Data expansion must take place on the satellite product before it can be displayed. Normally this data expansion takes place in SATDATA, but since all field sites do not have access to a Hayes compatible modem required by the SATDATA program, SATEXPND is available. SATEXPND reads in the compressed data, expands the data out in memory, and writes the expanded data out to disk. Procedures for using the SATEXPND program are found in Appendix B: Alternative Communication Procedures.

```
/*      date:  May 1987
        version:  1.0
        by:  Glen W. Sampson, NWS WRH SSD

Purpose:
  The purpose of this program is to expand a compressed satellite data file,
so the display of a picture can be done by simply reading the data file from
disk.  This greatly reduces the display time.

Exits:
  The normal exit is by selecting the Return to DOS option.  If errors occur
while doing the disk I/O, this routine will exit with an error message.

Created by:
        (version 3.00 of the Lattice C Compiler)
        C>lc -cu -md satexpnd
        C>link satexpnd func cd, satexpnd, con, lcd
*/

/* define the parameters */
#define ERROR           -1              /* DOS error */
#define vert_ln         640             /* number of vertical lines in a display */
#define horz_ln         400             /* number of horizontal lines in a display */
#include "fcntl.h"
int prodflg;                            /* variables for func.asm */
int tx_ptr;
int optr;

void main()
{
    int i, j, k;                                    /* general counters */
    int fd_in, fd_out;                              /* file descriptors */
    int num;                                        /* run length number */
    int val;                                        /* run length value */
    long int size;                                  /* size of the input file */
    char hdr[20];                                   /* file header array */
    char s[128];                                    /* stdin string */
    char str[128];                                  /* secondary stdin string */
    char *max_in;                                   /* end of the input buffer */
```

```
char *buf, *buf_out, *buf_out1, *buf_out2;          /* output buffer pointers */
char *buf_in;                                       /* input buffer pointer */
char *ptr_sv;                                       /* variable to save a pointer value */
char *getml();                                      /* declare pointer function */

/* init screen and output buffer memory */
clear();
if ((buf_out = getml (128000L)) == 0)
    fatal ("ERROR:  allocating output file memory^G");


/* loop until the user has had his fill of this routine */
for(;;) {

    /* get the filename of the compressed data */
    for(;;) {
            printf("\nEnter the compressed data filename:  ");
            gets(&s[0]);                            /* get the user's response */
            fd_in = dopen (&s[0], O_RDONLY);        /* open the file */
            if (fd_in == ERROR) {
                    printf("ERROR IN OPENING FILE, TRY AGAIN^G\n");
                    continue;
            }
            break;
    }

    /* get the output filename for the expanded data */
    for(;;) {
            printf("Enter the output data filename:  ");
            gets(&s[0]);                                    /* get the user's response */
            fd_out = dopen (&s[0], O_WRONLY);               /* try to open the file */
            if (fd_out != ERROR) {
                    printf("  FILE ALREADY EXISTS, OVERWRITE (Y/N)?  ");
                    gets(&str[0]);                          /* to overwrite or not */
                    if (toupper(str[0]) == 'Y') {           /* yes - overwrite it */
                            dclose (fd_out);                /* close previous file first */
                            fd_out = dcreat (&s[0], 0);     /* clear out the old file */
                    }
                    else                                    /* do not overwrite */
                            continue;
            }
            else
                    fd_out = dcreat (&s[0], 0);             /* create a new file */
            break;
    }

    /* read/write the file headers */
    printf("Now processing the data\n");            /* if user of progress */
    if (dread (fd_in, &hdr[0], 20) == ERROR)
            fatal ("ERROR:  reading input file header^G");
    printf("processing:  %c%c%c%c_%c%c%c%c%c%c_",hdr[2],hdr[3],hdr[4],hdr[5],hdr[6],hdr[7],hdr[8],hdr[9],hdr[10
```

```
printf("%c%c_%c%c%c",hdr[12],hdr[13],hdr[14],hdr[15],hdr[16]);
if (Dwrite (fd_out, &hdr[0], 20) == ERROR)
        fatal ("ERROR:  writing output file header^G");
size = hdr[1] * 256L + hdr[0];                  /* calculate input file size */

/* allocate input memory and read data */
if ((buf_in = getml(size)) == 0)
        fatal ("ERROR:  allocating input file memory^G");
if (dread (fd_in, buf_in, size) == ERROR)
        fatal ("ERROR:  reading in compressed data^G");

/* expand the data and write to the ouput file */
max_in = buf_in + size;                         /* calculate the end of the input buffer */
ptr_sv = buf_in;                                /* save our current pointer */
buf_out1 = buf_out;                             /* init output buffer pointers */
buf_out2 = buf_out + 64000L + vert_ln/2;
buf = buf_out1;
for (i = 0; i < horz_ln && buf_in < max_in; i++) {
        num = *buf_in & 15;                     /* isolate the run length */
        for (j = 0; j < vert_ln && num > 0;) {  /* expand the data */
                val = *buf_in >> 4;             /* isolate the value */
                for (k = num; k > 0 && j < vert_ln; k--, j++)
                        buf[j] = val;
                buf_in++;                       /* update the pointer */
                num = *buf_in & 15;             /* get the next run length */
        }
        if (i % 2) {                            /* determine the proper output buffer */
                pack (buf_out2);                /* pack output buffer */
                buf_out2 += vert_ln/2;          /* update the pointer */
                buf = buf_out1;                 /* switch buffers */
        }
        else {
                pack (buf_out1);
                buf_out1 += vert_ln/2;
                buf = buf_out2;
        }
        buf_in++;                               /* skip the two nulls at the end */
        if (*buf_in++ != 0) {
                printf("DATA ERROR IN LINE %d^G\n", i);
                while (*buf_in != 0 && buf_in < max_in)
                        buf_in++;               /* find the end of a line */
                buf_in += 2;                    /* skip the two nulls */
        }
}
/* write the data out to disk */
buf = buf_out + 64000L;                         /* find the end of first pix buffer */
for (; buf_out1 < buf;)                         /* fill remainder of file with nulls */
        *buf_out1++ = 0;
buf = buf_out + 128000L + vert_ln/2;
for (; buf_out2 < buf;)
        *buf_out2 = 0;
buf_out1 = buf_out;                             /* reset pointer to the beginning of the buffer */
```

```
        if (dwrite (fd_out, buf_out1, (unsigned) 64000) == ERROR)
                fatal ("ERROR:  writing the first output buffer to disk^G");
        buf_out2 = buf_out + 64000L + vert_ln/2;        /* reset second pointer to the beginning */
        if (dwrite (fd_out, buf_out2, (unsigned) 64000) == ERROR)
                fatal ("ERROR:  writing the second output buffer to disk^G");
        dclose (fd_in);                                 /* close all files */
        dclose (fd_out);
        rlsml (ptr_sv, size);                           /* release the input memory */
        printf("Do you want to expand another file (Y/N)?  ");
        gets(&str[0]);                                  /* get the user response */
        if (toupper (str[0]) == 'N')
                break;                                  /* exit this loop */
    }
    rlsml (buf_out, 128000L);                           /* release output memory */
    exit();                                             /* return to DOS */
}
/*****************************************************************************/
/*                                                                           */
/*                           F U N C T I O N S                               */
/*                                                                           */
/*****************************************************************************/

/*----------------------------< start of fatal >-----------------------------*/
/*****  name:  fatal (&string)
purpose:  This routine prints an error message on the screen and returns
          to DOS.
callers:  main()
global variables used:  none
*/
fatal (string)
char *string;
{
        printf("%s", string);                           /* display the error message */
        exit();                                         /* return to DOS */
        return(0);
}
/*----------------------------< end of fatal >-------------------------------*/

/*----------------------------< start of pack >------------------------------*/
/*****  name:  pack (&buffer)
purpose:  This routine takes two nibbles and stuffs them into one byte.
callers:  main()
global variables used:  none
*/
pack (buffer)
char buffer[];
{
        int i, j;                                       /* general counters */

        for (i = 0, j = 0; j < vert_ln; i++, j++)
                buffer[i] = (buffer[j++] << 4) | buffer[j];
        return(0);                                      /* all stuffed in */
```

```
}
/*----------------------------< end of pack >---------------------------------*/
```

V. HARDWARE DOCUMENTATION

This chapter describes the basic hardware configuration used in the entire system. Complete instructions on assembling the signal demodulator are included. All of the cabling used to connect the various computers together and telephone numbers of the hardware suppliers are described in Appendix C, Miscellaneous Hardware Information.

A. General Hardware Description

The A/D converter module consists of the following hardware with the approximate costs:

IBM PC, 256 kb memory, 1 floppy drive	$840
Data Translation DT2814 (A/D converter)	$395
Monochrome monitor and adapter	$320
Signal demodulator	$150
Cabling	$50
TOTAL	$1750

Instructions for assembling the signal demodulator and the associated documentation are found in section B of this chapter, Signal Demodulator.

The dissemination hardware is intended to be any computer system with the required capabilities. The general hardware on the system used in Western Region follows:

Data General S230 Eclipse, 512 kb memory
ALM 8 (quantity of three)
Racal Vadic VA1616/80 chassis
Racal Vadic VA3480 modem
Racal Vadic VA2010 power supply
System Industries 160 MB hard disk

Other peripherals are also contained on this system, but they are not pertinent to this project.

The remotely located display system consists of the following hardware:

IBM PC, 256 kb memory, 2 floppy drives	$930
Hayes 1200B Modem (modem only)	$250
Standard Color Monitor	$410
Tecmar Graphics Master adapter	$450
TOTAL	$2040

The amount of memory directly determines the number of satellite pictures which can be animated. The base configuration of 256 kb will allow 3 pictures to be animated. Increasing the memory to capacity (640 kb) will allow 9 pictures to be animated.

B. Signal Demodulator

The signal demodulator was designed to meet six major criteria in order to serve as a viable interface between the dedicated GOESTAP telephone line and the IBM PC computer. The circuit diagram in Figure 5 depicts how the following six design objectives were accomplished.

NOTE: In the following, U1, U2 and U3 refer to integrated chip 1/4ECG859. A1 refers to J-FET ECG451. CR1 and CR2 refer to IN914 diodes. CR3 and CR4 refer to ECG519.

1. Conditioning of the input GOESTAP signals.

   The dedicated GOESTAP telephone line is connected to input transformer T1 which provides telephone line impedance matching and coupling for the interface circuitry. R1 provides secondary winding impedance matching.

2. Automatic gain control.

   Compensation for unstable input line levels is accomplished by use of an automated gain control. Line level information is determined from the maximum modulation found in the scan sync at the beginning of every line. This sync occurs every 0.5 seconds, and is stored by C1. This stored charge applied to the gate of A1 will set the level of input attenuation. The RC time constant resulting from the capacitance of C1 and the resistance of R46 determine the "roll off" period, or the amount of time before the next sync update is needed. These two values must be balanced for maximum AGC agility and linearity throughout the 0.5 second display line period. An attack time of 6 msec. and decay time of 12 seconds appears to work best.

   The GOESTAP signal available at U1 pin 1 will then be uniform from line to line, and tolerant of up to a ±10dB swing from the nominal -16dB dedicated telephone levels. The center of the operating range of this circuit is determined by the gain of U1-A. A 10K resistor, not 39K, must be used for R19 on 0dBM GOESTAP drops.

3. Balancing and full wave rectification of the GOESTAP signals.

   The amplitude modulated (AM) signal is sent to an absolute value circuit consisting of U1-A, U-1B, CR1, CR2, and associated circuitry. Here, it is full wave rectified. The null offset circuitry connected to U1-A pin 3 allows phase balance calibrations to be performed. U1-A does the actual wave shaping; U1-B serves merely as a buffer

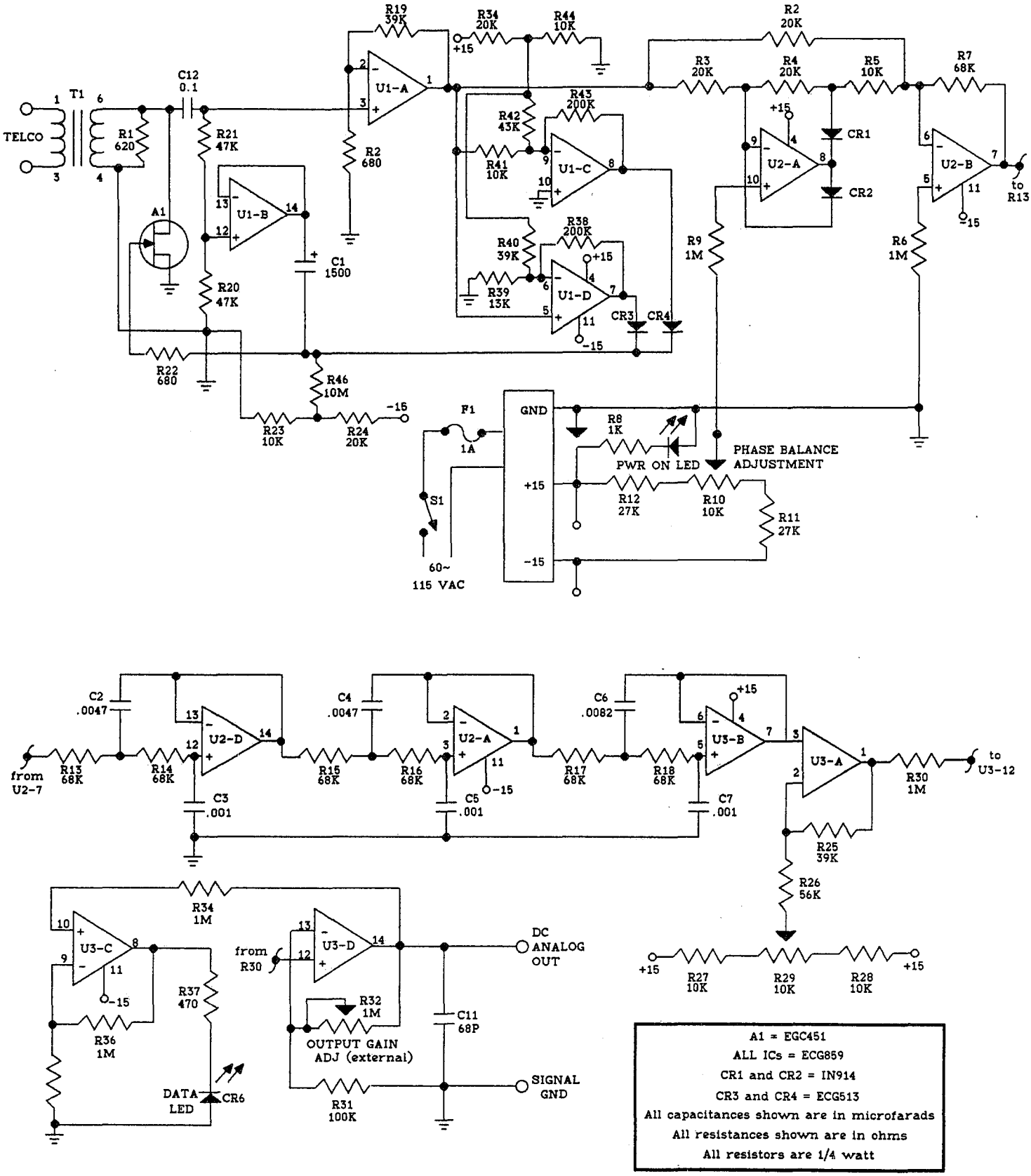


Figure 5 - Signal demodulator circuit diagram

amplifier to the next function which is the removal of the 2400 Hz GOESTAP carrier.

4. Removal of the GOESTAP 2400 Hz carrier.

   An analog base band signal is derived from the rectified, AM GOESTAP signal through the use of a 6th order low pass filter. U2-B, U2-A, U3-B, and associated circuitry act as successive stages of the filtering network. Thus, the carrier is removed, and the resultant DC analog (available for use at U3-B pin 7) has a 20 microsecond rise and fall time with less than 4% residual carrier ripple.

5. Development of a high impedance analog baseband output.

   The signal demodulator output null circuit and output driver, U2-C and U3-D, allow calibration of output null and gain through R29 and R32 respectively. Zero VDC (during sync) to +5VDC (for maximum modulation) are recommended; however, a linear scale through ±10VDC is possible. The remaining op amp stage U3-C merely drives a light emitting diode to indicate that data is being processed.

6. Low in cost, high in reliability, and ease of operation.

   The signal demodulator uses +15VDC, -15VDC and ground. It is powered by an Acopian model #15E3/D AC to DC power module fused at 0.5 Amp and switch connected to 115VAC. A light emitting diode indicates when the +15VDC power form is present.

   The analog output gain is the only external adjustment, and requires minimal operator intervention once initially set.

C. Signal Demodulator Alignment Procedures

The following procedure should be used to calibrate the signal demodulator:

1. Connect the signal demodulator to your designated GOESTAP dedicated telephone line using the two telco input lugs. Polarity is unimportant here.

2. Connect channel one of an oscilloscope to the analog output lug grounding it to the remaining (ground) lug. These should already be connected to the PC. Polarity is important here.

3. Set the oscilloscope as follows:
   volts/div = 1
   time/div = 10 msec.
   trigger mode = auto
   trigger source = line
   vertical mode = channel 1

4. During the receiver phasing (black) portion of the GOESTAP signal, which occurs at about 10 seconds into the transition, adjust R32 on the front of the box for a maximum level of +5VDC.

5. Adjust R29 (output null) for 0VDC during the scan line sync found at the beginning of every data line which occurs every 0.5 seconds.

6. Adjust R10 (phase balance) for minimum ripple on the signal. This adjustment is done best by changing the oscilloscope time/div to 2 msec.

7. These three adjustments are interdependent. Perform steps 4, 5, and 6 several times until all criteria are met.

VI. REFERENCES

Berlin, Howard M., 1977: Design of Active Filters, with Experiments. Howard W. Sams and Company, Inc., First Edition.

Buschbaum, Walter H., 1978: Buschbaum's Complete Handbook of Practical Electronic Reference Data. Prentice Hall, Second Edition.

The GOES User's Guide. Department of Commerce, June 1983, pp 6-29 to 6-33.

Graphics Master Technical Reference. Tecmar Presses #931661 Revision A, 1984.

Hill, Norman M., 1987: Audio AGC Has 40dB Dynamic Range. Design Ideas EDN, Volume 32, Number 15, July 23, 1987, Cahners Publishing Company.

IBM Technical Reference: Personal Computer XT. #1502237, April 1983.

Jung, Walter G., 1986: IC Op-Amp Cookbook. Howard W. Sams and Company, Third Edition.

User Manual for DT2814. Data Translation UM-05143-0, October 1985.

Willen, D. and J. Krantz, 1983: 8088 Assembler Language Programming: The IBM PC. Howard W. Sams and Company, Second Edition.

APPENDIX A: SATELLITE DATA FORMAT

Satellite data can exist in two formats: expanded and compressed. Each of these formats contains a 20 byte header at the beginning of the data. The header has the format:

compressed file size - 2 bytes (low value, high value)
picture UTC time - 4 bytes (hhmm)
picture UTC date - 6 bytes (DDMMYY)
picture enhancement curve - 2 bytes
picture sector information - 3 bytes
unused - 3 bytes (generally nulls)

After the above header, the compressed data contains a pixel value and a run length count in each byte. The first four bits represent the pixel value, and the next four bits represent the number of times the pixel value should be repeated in the display line. Each line of data is separated by two nulls; thus error checking can occur on the data because 640 pixel values (one display line) must always be separated by two nulls. If this condition does not occur at the receiving site, an error has occurred in the transmission process.

The expanded data is considerably larger than the compressed data with the file size being 128020 bytes. After the file header, the next 64000 bytes contain the odd numbered display lines (i.e. 1,3,5,...,399), and the next 64000 bytes contain the even numbered display lines (i.e. 0,2,4,...,398). Each display line consists of 320 bytes with each byte representing two pixel values (4 bits per pixel value). The display lines are not separated by special characters as in the compressed format, and must be determined by file location. Data output from the SATDATA program is in the expanded format, although the actual data transmission occurs in the compressed format.

APPENDIX B: ALTERNATIVE COMMUNICATION PROCEDURES

If a site does not have access to a Hayes compatible modem, other communication alternatives are available. First, any off-the-shelf communication hardware/software package can be used that has the ability to receive a binary file and store that file on disk. After the data resides on disk, the SATEXPND program must be run to expand the data for display. The SATEXPND program prompts the user for all information, so answer the questions as they appear on the display.

Another alternative exists using the SATDATA program. Since only the autodial portion of the SATDATA program requires a Hayes compatible modem, this portion of the program must be disabled. When executing the SATINFO program, if no telephone number is given (i.e. the return key is entered with no data) SATDATA will assume a connection already exists with the dissemination computer system. Therefore, a modem connection can be established before SATDATA is executed, and the Hayes compatibility problem is circumvented.

APPENDIX C: MISCELLANEOUS HARDWARE INFORMATION

A. Pertinent Telephone Numbers and Addresses

IBM analog to digital converter hardware (DT2814):
Data Translation Inc.
100 Locke Drive
Marlboro, MA 01752
(617) 481-3700

B. Signal Demodulator Parts List

The following is a list of components necessary to construct this signal demodulator.

i. Chassis hardware and electrical components
1. Project box (3"x6"x8" approximately)
2. Terminal strip (4 lug minimum)
3. Fuse holder (3AG)
4. 0.5A fuse (3AG)
5. Miniature SPDT switch (2A minimum)
6. Line cord (molded, with ground)
7. Stand-offs, screws, and nuts (4 each)
8. 3 feet stranded AWG22 wire
9. Volume knob

ii. Miscellaneous electronic components
1. ±15VDC power supply with ground
2. 2 each light emitting diodes
3. 3 each ECG859 quad op amp
4. 3 each 14 pin dip sockets
5. 2 each ECG519 diodes
6. 1 - ECG451 transistor
7. 1 Triad SP-67 audio transformer
8. 2 each IN914 diodes
9. 1 - printed circuit board

iii. Potentiometers
1. 1 linear taper, chassis mount, 1 meg.
2. 2 each multi-turn, PC mount, 10K

iv. Fixed carbon resistors (1/4 watt, 10%)
1. 2 each 4K ohm
2. 1 - 620-ohm
3. 1 - 1K-ohm
4. 6 each 10K ohm
5. 5 each 20K ohm
6. 2 each 27K ohm
7. 2 each 680 ohm
8. 7 each 68K ohm
9. 1 - 10M ohm
10. 1 - 43K ohm

11. 2 each 39K ohm
12. 1 - 13K ohm
13. 2 each 200K ohm
14. 3 each 100K ohm
15. 1 - 470 ohm
16. 4 each 1M ohm

v. Capacitors (>= 15VDC, 10%)

1. 1 - 68 pf
2. 3 each 0.001 uf
3. 2 each 0.0047 uf
4. 1 - 0.0082 uf
5. 1 - 1500 uf
6. 1 - 0.47 uf
7. 1 - 0.1 uf

C. Cabling Information

Two cables are used in the current configuration. A cable from the signal demodulator to the DT2814 digitizer, and from the IBM PC to the dissemination computer. Since the dissemination computer is likely to change from installation to installation, only the signal demodulator to the DT2814 cable is described here.

Input to the DT2814 is thru a 20 pin "card edge" connector with a ribbon cable, and output from the demodulator is on a screw terminal. Pin 1 of the ribbon cable should be connected to the signal demodulator output, and pin 18 should be connected to the demodulator signal ground.

D. IBM Card Settings

i. DT2814 - A/D converter board

The DT2814 is setup with a base address of 2F8H, input range of 0 to +5VDC, base frequency of 300KHz and interrupt IRQ3. This configuration corresponds to the following jumper installation:

W1 - jumpered
W2 - jumpered
W3 to W9 - not jumpered
W10 - jumpered
W11 - not jumpered
W12 - jumpered
W13 to W16 - not jumpered
W17 - jumpered
W18 - not jumpered
W19 - not jumpered
W20 - jumpered

ii. Tecmar Graphics Master

JPR 1 - positions A, B, and C are jumpered; D, E, and F are not jumpered
JPR 3 - do not change (should not be jumpered)
JPR 4 - jumpered
JPR 5 - middle two pins are jumpered
JPR 6 - jumpered
JPR 7 - not jumpered
JPR 8 - not jumpered
JPR 10 - do not change (should not be jumpered)
JPR 11 - do not change (should not be jumpered)

SW 1 - should be down for a color monitor

The system board switch of the PC must also be configured in an 80 column mode for a color monitor. Please refer to the IBM Guide to Operations manual for the correct switch setting.

APPENDIX D: EXAMPLE OF DISSEMINATION COMPUTER IMAGE LISTING

When a field site dials into dials into the dissemination computer, they receive a listing of available images. The total number of pictures available is forty, although this number is only determined by the number of picture titles which can conveniently be displayed on the IBM screen. A sample listing follows:

Picture currently available are:

1.	2031_10SE87___EB2	21.	1445_10SE87_ZA_WD1
2.	2015_10SE87_C4_SB1	22.	1445_10SE87_C4_SB1
3.	2001_10SE87_ZA_EC1	23.	1431_10SE87___EB2
4.	1945_10SE87_C4_SB1	24.	1415_10SE87_HF_WC1
5.	1931_10SE87___EB2	25.	1401_10SE87_ZA_EC1
6.	1915_10SE87_C4_SB1	26.	1345_10SE87_HF_WB1
7.	1845_10SE87_C4_SB1	27.	1331_10SE87___EB2
8.	1831_10SE87___EB2	28.	1245_10SE87_HF_WB1
9.	1815_10SE87_C4_SB1	29.	1231_10SE87___EB2
10.	1745_10SE87_ZA_WD1	30.	1215_10SE87_HF_WC1
11.	1745_10SE87_C4_SB1	31.	1145_10SE87_ZA_WD1
12.	1731_10SE87___EB2	32.	1145_10SE87_HF_WB1
13.	1715_10SE87_ZD_WC1	33.	1131_10SE87_MB_EB2
14.	1701_10SE87_ZA_EC1	34.	1115_10SE87_ZD_WC1
15.	1645_10SE87_C4_SB1	35.	1101_10SE87_ZA_EC1
16.	1631_10SE87___EB2	36.	1015_10SE87_ZA_WD1
17.	1601_10SE87_ZA_EC1	37.	1015_10SE87_HF_WC1
18.	1545_10SE87_C4_WA2	38.	1001_10SE87_ZA_EC1
19.	1531_10SE87___EB2	39.	0931_10SE87_MB_EB2
20.	1515_10SE87_C4_SB1	40.	0831_10SE87_MB_EB2

Enter the picture number, or zero for no picture -

APPENDIX E: SATDSP MENU DESCRIPTIONS

A brief description of the four menus found in the SATDSP program is given here. The menus are relatively easy to use and are specifically designed for a person to use with no additional instructions. Upon entering SATDSP the main menu appears.

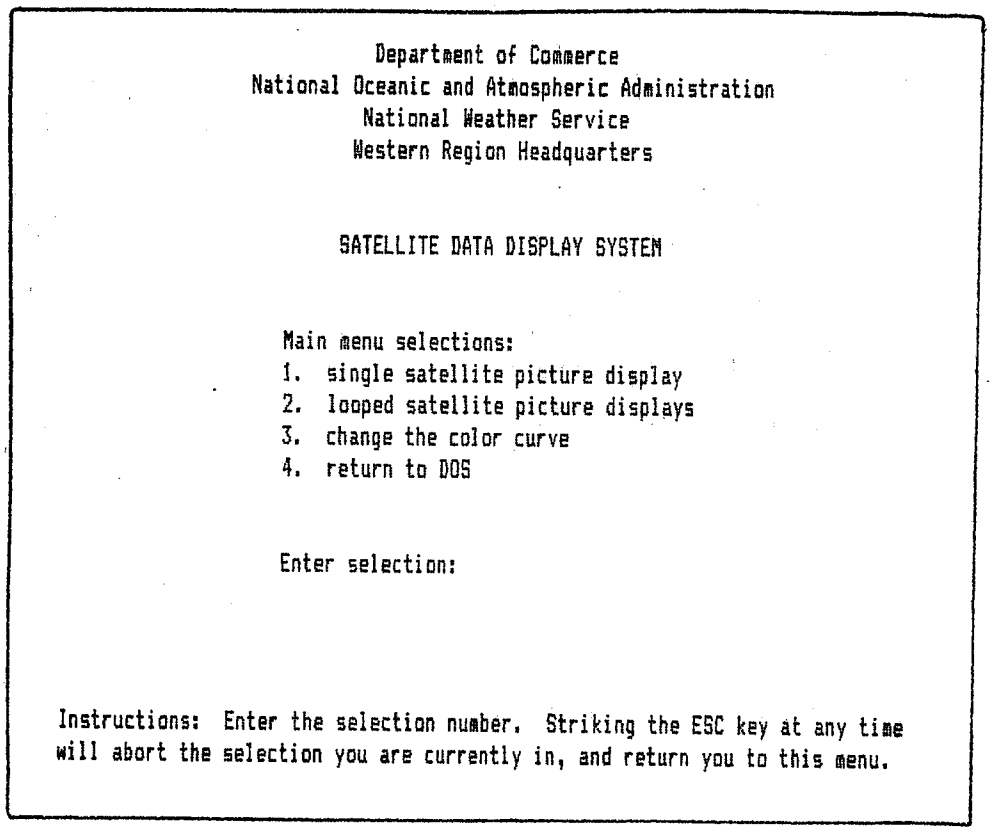


Figure 6 - The main menu in SATDSP. The most important instruction for the user to remember here is that the ESC key will always the program to this display.

The main menu provides easy access to the four options provide by the program. Option 1 (single satellite picture display) allows the user to display a single image, option 2 (looped satellite picture displays) provides for the animation of images, option 3 (change the color curve) allows a user to edit the existing curve or retrieve a previously stored curve, and option 4 (return to DOS) enables the user to exit SATDSP in an orderly manner. Examples of these additional displays are given in Figures 7 through 9.

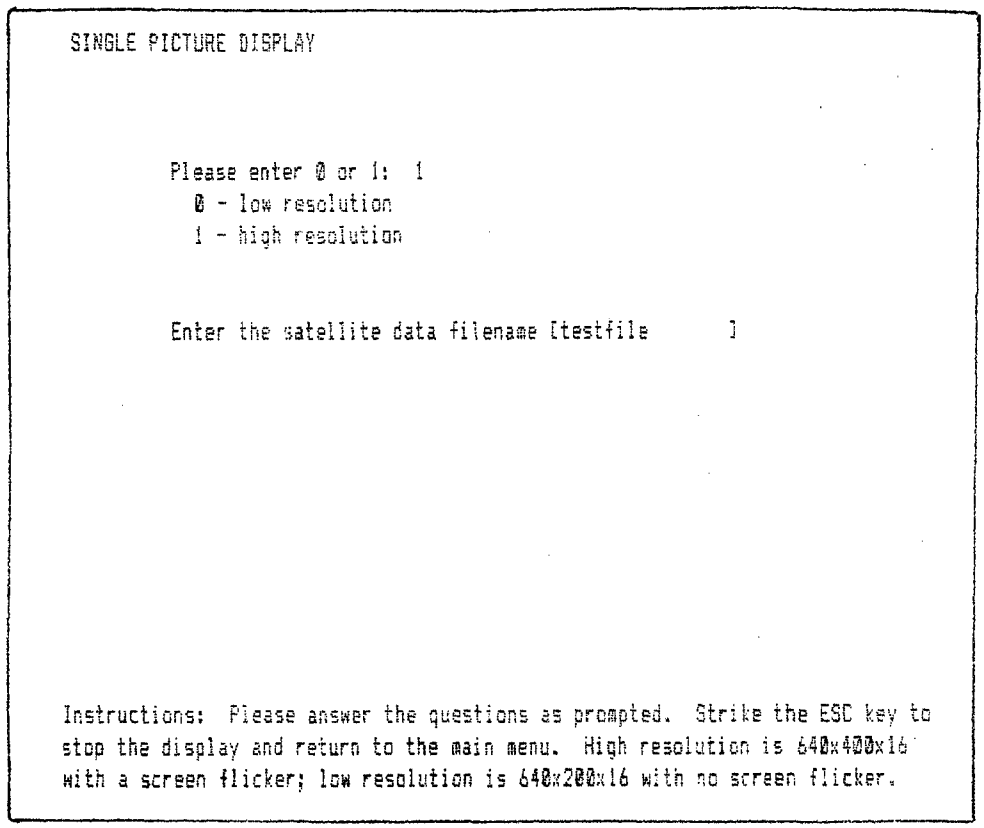


Figure 7 - Single image display menu. A user can display a single image with this menu option. The high resolution display doubles the data resolution via an interlace technique. Thus the high resolution display has a slight flicker if a long persistent phosphor monitor is not used.

139 Aids for Forecasting Minimum Temperature in the Wenatchee Frost District. Robert S. Robinson, April 1979. (PB298339/AS)
140 Influence of Cloudiness on Summertime Temperatures in the Eastern Washington Fire Weather district. James Holcomb, April 1979. (PB298674/AS)
141 Comparison of LFM and MFM Precipitation Guidance for Nevada During Doreen. Christopher Hill, April 1979. (PB298613/AS)
142 The Usefulness of Data from Mountaintop Fire Lookout Stations in Determining Atmospheric Stability. Jonathan W. Corey, April 1979. (PB298899/AS)
143 The Depth of the Marine Layer at San Diego as Related to Subsequent Cool Season Precipitation Episodes in Arizona. Ira S. Brenner, May 1979. (PB298817/AS)
144 Arizona Cool Season Climatological Surface Wind and Pressure Gradient Study. Ira S. Brenner, May 1979. (PB298900/AS)
146 The BART Experiment. Morris S. Webb, October 1979. (PB80 155112)
147 Occurrence and Distribution of Flash Floods in the Western Region. Thomas L. Dietrich, December 1979. (PB80 160344)
149 Misinterpretations of Precipitation Probability Forecasts. Allan H. Murphy, Sarah Lichtenstein, Baruch Fischhoff, and Robert L. Winkler, February 1980. (PB80 174576)
150 Annual Data and Verification Tabulation - Eastern and Central North Pacific Tropical Storms and Hurricanes 1979. Emil B. Gunther and Staff, EPHC, April 1980. (PB80 220486)
151 NMC Model Performance in the Northeast Pacific. James E. Overland, PMEL-ERL, April 1980. (PB80 196033)
152 Climate of Salt Lake City, Utah. Wilbur E. Figgins, Third Revision January 1987. (PB87 157194/AS)
153 An Automatic Lightning Detection System in Northern California. James E. Rea and Chris E. Fontana, June 1980. (PB80 225592)
154 Regression Equation for the Peak Wind Gust 6 to 12 Hours in Advance at Great Falls During Strong Downslope Wind Storms. Michael J. Oard, July 1980. (PB91 108367)
155 A Raininess Index for the Arizona Monsoon. John H. Ten Harkel, July 1980. (PB81 106494)
156 The Effects of Terrain Distribution on Summer Thunderstorm Activity at Reno, Nevada. Christopher Dean Hill, July 1980. (PB81 102501)
157 An Operational Evaluation of the Scofield/Oliver Technique for Estimating Precipitation Rates from Satellite Imagery. Richard Ochoa, August 1980. (PB81 108227)
158 Hydrology Practicum. Thomas Dietrich, September 1980. (PB81 134033)
159 Tropical Cyclone Effects on California. Arnold Court, October 1980. (PB81 133779)
160 Eastern North Pacific Tropical Cyclone Occurrences During Intraseasonal Periods. Preston W. Leftwich and Gail M. Brown, February 1981. (PB81 205494)
161 Solar Radiation as a Sole Source of Energy for Photovoltaics in Las Vegas, Nevada, for July and December. Darryl Randerson, April 1981. (PB81 224503)
162 A Systems Approach to Real-Time Runoff Analysis with a Deterministic Rainfall-Runoff Model. Robert J.C. Burnash and R. Larry Ferral, April 1981. (PB81 224495)
163 A Comparison of Two Methods for Forecasting Thunderstorms at Luke Air Force Base, Arizona. LTC Keith R. Cooley, April 1981. (PB81 225393)
164 An Objective Aid for Forecasting Afternoon Relative Humidity Along the Washington Cascade East Slopes. Robert S. Robinson, April 1981. (PB81 23078)
165 Annual Data and Verification Tabulation, Eastern North Pacific Tropical Storms and Hurricanes 1980. Emil B. Gunther and Staff, May 1981. (PB82 230336)
166 Preliminary Estimates of Wind Power Potential at the nevada Test Site. Howard G. Booth, June 1981. (PB82 127036)
167 ARAP User's Guide. Mark Mathewson, July 1981, revised September 1981. (PB82 196783)
168 Forecasting the Onset of Coastal Gales Off Washington-Oregon. John R. Zimmerman and William D. Burton, August 1981. (PB82 127051)
169 A Statistical-Dynamical Model for Prediction of Tropical Cyclone Motion in the Eastern North Pacific Ocean. Preston W. Leftwich, Jr., October 1981. (PB82195298)
170 An Enhanced Plotter for Surface Airways Observations. Andrew J. Spry and Jeffrey L. Anderson, October 1981. (PB82 153883)
171 Verification of 72-Hour 500-MB Map-Type Predictions. R.F. Quiring, November 1981. (PB82 158098)
172 Forecasting Heavy Snow at Wenatchee, Washington. James W. Holcomb, December 1981. (PB82 177783)
173 Central San Joaquin Valley Type Maps. Thomas R. Crossan, December 1981. (PB82 196064)
174 ARAP Test Results. Mark A. Mathewson, December 1981. (PB82 198103)
176 Approximations to the Peak Surface Wind Gusts from Desert Thunderstorms. Darryl Randerson, June 1982. (PB82 253089)
177 Climate of Phoenix, Arizona. Robert J. Schmidli, April 1969 (revised December 1986). (PB87 142063/AS)
178 Annual Data and Verification Tabulation, Eastern North Pacific Tropical Storms and Hurricanes 1982. E.B. Gunther, June 1983. (PB85 106078)
179 Stratified Maximum Temperature Relationships Between Sixteen Zone Stations in Arizona and Respective Key Stations. Ira S. Brenner, June 1983. (PB83 249904)
180 Standard Hydrologic Exchange Format (SHEF) Version I. Phillip A. Pasteries, Vernon C. Bissel, David G. Bennett, August 1983. (PB85 106052)
181 Quantitative and Spacial Distribution of Winter Precipitation along Utah's Wasatch Front. Lawrence B. Dunn, August 1983. (PB85 106912)
182 500 Millibar Sign Frequency Teleconnection Charts - Winter. Lawrence B. Dunn, December 1983. (PB85 106276)
183 500 Millibar Sign Frequency Teleconnection Charts - Spring. Lawrence B. Dunn, January 1984. (PB85 111367)
184 Collection and Use of Lightning Strike Data in the Western U.S. During Summer 1983. Glenn Rasch and Mark Mathewson, February 1984. (PB85 110534)
185 500 Millibar Sign Frequency Teleconnection Charts - Summer. Lawrence B. Dunn, March 1984. (PB85 111359)
186 Annual Data and Verification Tabulation eastern North Pacific Tropical Storms and Hurricanes 1983. E.B. Gunther, March 1984. (PB85 109635)
187 500 Millibar Sign Frequency Teleconnection Charts - Fall. Lawrence B. Dunn, May 1984. (PB85 110930)
188 The Use and Interpretation of Isentropic Analyses. Jeffrey L. Anderson, October 1984. (PB85 132694)
189 Annual Data & Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1984. E.B. Gunther and R.L. Cross, April 1985. (PB85 1878887AS)
190 Great Salt Lake Effect Snowfall: Some Notes and An Example. David M. Carpenter, October 1985. (PB86 119153/AS)
191 Large Scale Patterns Associated with Major Freeze Episodes in the Agricultural Southwest. Ronald S. Hamilton and Glenn R. Lussky, December 1985. (PB86 144474AS)
192 NWR Voice Synthesis Project: Phase I. Glen W. Sampson, January 1986. (PB86 145604/AS)
193 The MCC - An Overview and Case Study on Its Impact in the Western United States. Glenn R. Lussky, March 1986. (PB86 170651/AS)
194 Annual Data and Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1985. E.B. Gunther and R.L. Cross, March 1986. (PB86 170941/AS)
195 Radid Interpretation Guidelines. Roger G. Pappas, March 1986. (PB86 177680/AS)
196 A Mesoscale Convective Complex Type Storm over the Desert Southwest. Darryl Randerson, April 1986. (PB86 190998/AS)
197 The Effects of Eastern North Pacific Tropical Cyclones on the Southwestern United States. Walter Smith, August 1986. (PB87 106258AS)
198 Preliminary Lightning Climatology Studies for Idaho. Christopher D. Hill, Carl J. Gorski, and Michael C. Conger, April 1987. (PB87 180196/AS)
199 Heavy Rains and Flooding in Montana: A Case for Slantwise Convection. Glenn R. Lussky, April 1987. (PB87 185229/AS)
200 Annual Data and Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1986. Roger L. Cross and Kenneth B. Mielke, September 1987.

NOAA SCIENTIFIC AND TECHNICAL PUBLICATIONS

The National Oceanic and Atmospheric Administration was established as part of the Department of Commerce on October 3, 1970. The mission responsibilities of NOAA are to assess the socioeconomic impact of natural and technological changes in the environment and to monitor and predict the state of the solid Earth, the oceans and their living resources, the atmosphere, and the space environment of the Earth.

The major components of NOAA regularly produce various types of scientific and technical information in the following kinds of publications:

PROFESSIONAL PAPERS — Important definitive research results, major techniques, and special investigations.

CONTRACT AND GRANT REPORTS — Reports prepared by contractors or grantees under NOAA sponsorship.

ATLAS — Presentation of analyzed data generally in the form of maps showing distribution of rainfall, chemical and physical conditions of oceans and atmosphere, distribution of fishes and marine mammals, ionospheric conditions, etc.

TECHNICAL SERVICE PUBLICATIONS — Reports containing data, observations, instructions, etc. A partial listing includes data serials; prediction and outlook periodicals; technical manuals, training papers, planning reports, and information serials; and miscellaneous technical publications.

TECHNICAL REPORTS — Journal quality with extensive details, mathematical developments, or data listings.

TECHNICAL MEMORANDUMS — Reports of preliminary, partial, or negative research or technology results, interim instructions, and the like.

Information on availability of NOAA publications can be obtained from:

ENVIRONMENTAL SCIENCE INFORMATION CENTER (D822)
ENVIRONMENTAL DATA AND INFORMATION SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
U.S. DEPARTMENT OF COMMERCE

6009 Executive Boulevard
Rockville, MD 20852