

McClellan AFB  
Contract #: F04606-85-C0733  
CDRL Item: E004  
24 May 90

**VERSION DESCRIPTION DOCUMENT**

**FOR THE**

**AN/FMQ-13(V)**

**DIGITAL INDICATOR/RECORDER**

**APPLICATION PROGRAM**

CPIN: 83M-FMQ13-F001-00A  
ENGR SPEC #: MMA 81-025B  
REVISION: INITIAL RELEASE

## SOFTWARE DELIVERY DOCUMENTATION

This document describes the media and format on which the source code and object code of the CPCI titled "Digital Indicator/Recorder Application Program", CPIN 83M-FMQ13-F001-00A is delivered. Both types of code are delivered on a 5.25 inch, double sided, double density floppy disk with a formatted capacity of 360K bytes.

The source code resides on one floppy disk. The disk is formatted into nine sectors per track with 40 tracks per side, yielding a total of 80 tracks. Each record is 512 bytes. Source code is accessed by block number using the target compiler disk "Digital Wind Measuring System Target Compiler", CPIN 83M-FMQ13-F003-00A.

The object code resides one floppy disk. The disk format is nine sectors per track with 40 tracks per side, yielding a total of 80 tracks. The record is MS-DOS compatible. The object code is provided in three different formats in three separate files. The file "IND-REC.BIN" is a memory image of the program in binary format. The file "IND-REC.HEX" provides the object code in Intel Hex Format. The file "IND-REC.ASC" is an ASCII listing of the object code.

These floppies should be protected from dust and contamination. The floppies should not be folded or otherwise deformed. Care should be taken to avoid exposure to intense magnetic fields.

## 1. INVENTORY

The following table provides a list all items covered by the CPCI number and version. In addition all utility and/or support computer program release documents which are not part of the released item but which are required to operate, load, or regenerate the released CPCI are listed with the quantity marked as "REF".

### INVENTORY LIST

ITEM	QTY	DESCRIPTION
1	2	DISK, 5.25" FLOPPY, PROGRAM SOURCE CODE
2	1	DISK, 5.25" FLOPPY, PROGRAM OBJECT CODE
3	1	LISTING, PAPER COPY, SOURCE CODE
4	1	LISTING, PAPER COPY, OBJECT CODE
5	3	SPECIFICATION, CPCI
6	3	DOCUMENT, SOFTWARE DELIVERY
7	3	DOCUMENT, VERSION DESCRIPTION
8	REF	COMPILER, "DIGITAL WIND MEASURING SYSTEM TARGET COMPILER" CPIN 83M-FMQ13-F003-00A
9	REF	MANUAL, "polyFORTH II REFERENCE MANUAL"
10	REF	MANUAL, "INTEL 8086 CPU Supplement to the polyFORTH II Reference Manual"

## 2. INVENTORY OF CPCI CONTENTS

The CPCI being released is titled " Digital Indicator/Recorder Application Program", CPIN 83M-FMQ13-F001-00A. This is the operational program that runs the ID-2408/FMQ-13(V) Indicator and the RO-588/FMQ-13(V) Recorder. This program resides in one 128Kbit (16K by 8 bits) EPROM. A description of the program, source code and object code listings, and charts of the program are documented in CPCI specification titled "Computer Program Specification for the AN/FMQ-13(V) Digital Indicator/Recorder Application Program" supplied with this release.

The object code is provided in three different formats in three separate files. The file "IND-REC.BIN" is a memory image of the program in binary format. The file "IND-REC.HEX" provides the object code in Intel Hex Format. The file "IND-REC.ACS" is an ASCII listing of the object code. A paper copy listing of the Intel Hex format and the ASCII format is provided with this release.

## 3. CLASS II CHANGES INSTALLED

This is the initial release of this program. As such it has no class II changes.

## 4. CLASS I CHANGES INSTALLED

This is the initial release of this program. As such it has no class I changes.

## 5. ADAPTATION DATA

This program does not require any adaptation data.

## 6. INSTALLATION INSTRUCTIONS

This is the operational program that runs the ID-2408/FMQ-13(V) Indicator and the RO-588/FMQ-13(V) Recorder. This program resides in one 128Kbit (16K by 8 bits) EPROM. The program is delivered with each indicator and recorder as firmware. This EPROM reside on a PCBA referred to as the microprocessor board (P/N 66733ASSY6461-1014). The EPROM is not a field replaceable item. If required for depot maintenance a new EPROM may be generated by copying a known good EPROM (established by the EPROM check sum). An EPROM may also be generated by down loading the object code, which is in INTEL HEX loading format, to an EPROM programmer.

McClellan AFB  
Contract #: F04606-85-C0733  
CDRL Item: E00D  
15 May 90

**COMPUTER PROGRAM SPECIFICATION**

**FOR THE**

**AN/FMQ-13(V)**

**DIGITAL INDICATOR/RECORDER**

**APPLICATION PROGRAM**

AUTHENTICATED BY \_\_\_\_\_

APPROVED BY \_\_\_\_\_

(PROCURING ACTIVITY)

(CONTRACTOR)

DATE \_\_\_\_\_

DATE \_\_\_\_\_



## TABLE OF CONTENTS

1. SCOPE .....	1
2. APPLICABLE DOCUMENTS .....	1
3. REQUIREMENTS .....	2
3.0 GENERAL DESCRIPTION .....	2
3.1 CPCI STRUCTURAL DESCRIPTION .....	3
3.2 FUNCTIONAL FLOW DIAGRAMS/CHARTS .....	4
3.2.1 Control Functional Flow Diagram .....	4
3.2.2 Data Flow Diagram .....	5
3.3 INTERFACES .....	5
3.3.1 The Indicator/Recorder Microprocessor System .....	5
3.3.1.1 Inter-Assembly Communication Interface .....	5
3.3.1.2 AWDS Interface .....	6
3.3.1.3 Front Panel Interface .....	6
3.3.1.4 Real Time Clock(RTC) Interface .....	9
3.3.1.5 Miscellaneous Control and Status (MCS) Register .....	9
3.3.1.6 Printer Data Output Interface .....	9
3.3.1.7 Watchdog Timer Interface .....	9
3.3.2 Inter-Assembly Communication Protocol .....	11
3.3.2.1 Inter-Assembly Communication Message Format .....	11
3.3.2.1.1 Inter-Assembly Communication Opcode .....	11
3.3.2.1.2 Inter-Assembly Communication Message Data .....	11
3.3.2.1.3 Inter-Assembly Communication Message Timing .....	13
3.3.3 AWDS Interface Message and Protocol .....	13
3.4 PROGRAM INTERRUPTS .....	14
3.4.1 Inter-Assembly Receiver Interrupt .....	14
3.4.2 Inter-Assembly Communication Transmitter Interrupt .....	14
3.4.3 AWDS Receiver Interrupt .....	14
3.4.4 AWDS Transmitter Interrupt .....	14
3.4.5 Inter-Assembly External Status Interrupt .....	17
3.4.6 AWDS External/Status Interrupt .....	17
3.4.7 Special Receive Condition Interrupts .....	17
3.5 TIMING AND SEQUENCING DESCRIPTION .....	17
3.6 SPECIAL CONTROL FEATURES .....	17
3.7 STORAGE ALLOCATION .....	18
3.7.1 Data Base Definition .....	18
3.7.1.1 File Description .....	18
3.7.1.2 Table Description .....	18
3.7.1.3 Item Description .....	18
3.7.1.4 Graphic Table Description .....	18
3.7.1.5 CPCI Constants .....	18
3.7.2 CPC Relationship .....	20
3.8 OBJECT CODE CREATION .....	20

3.9 ADAPTATION DATA .....	20
3.10 DETAIL DESIGN DESCRIPTION .....	20
3.10.1 Identification of CPC No. 1 .....	20
3.10.1.1 Description of CPC No. 1 .....	20
3.10.1.1.1 Minute Interrupt .....	21
3.10.1.1.2 Second Interval Interrupt .....	21
3.10.1.1.3 100 MS Interrupt Interval .....	21
3.10.2 Identification of CPC No. 2 .....	21
3.10.2.1 CPC No. 2 Description of Modes of Operation .....	21
3.10.2.1.1 CPC No. 2 Description of Master Mode .....	22
3.10.2.1.1.1 CPC No. 2 Sensor Polling Sequence .....	22
3.10.2.1.2 CPC No. 2 Description of Normal Mode .....	22
3.10.2.1.3 CPC No. 2 Description of Backup Mode .....	22
3.10.2.2 Detailed Description of the Inter-Assembly Task .....	23
3.10.2.2.1 Description of Inter-Assembly Task Figure 10 .....	23
3.10.2.2.2 Description of Inter-Assembly Task Figure 11 .....	23
3.10.2.2.3 Description of Inter-Assembly Task Figure 12 .....	23
3.10.2.2.4 Description of Inter-Assembly Task Figure 13 .....	24
3.10.2.2.5 Description of Inter-Assembly Task Figure 14 .....	24
3.10.2.2.6 Description of Inter-Assembly Task Figure 15 .....	24
3.10.2.3 Detailed Description of the Inter-Assembly Communication Interrupt Handler .....	25
3.10.2.3.1 Interrupt by Carrier Status Change .....	25
3.10.2.3.2 Transmit Buffer Empty Interrupt .....	25
3.10.2.3.3 Received Character Interrupt .....	25
3.10.3 Identification of CPC No. 3 .....	26
3.10.3.1 CPC No. 3 Major Loop .....	26
3.10.3.2 CPC No. 3 Wind Data Processing .....	26
3.10.3.2.1 Direction Reference .....	27
3.10.3.2.1.1 Five Second Vectorial Average .....	27
3.10.3.2.1.2 Two and Ten Minute Vectorial Average .....	27
3.10.3.2.2 Gust Spread .....	27
3.10.3.2.3 Gusts .....	27
3.10.3.2.3.1 Gust Using Two-Minute Average Speed .....	28
3.10.3.2.3.2 Gust Using Ten-Minute Average Wind Speed .....	28
3.10.3.2.4 Peak Wind Calculations .....	28
3.10.3.2.5 Directional Variability .....	28
3.10.3.2.6 Standard Deviation of Direction .....	29
3.10.4 Identification of CPC No. 4 .....	29
3.10.4.1 CPC No. 4 Description .....	29
3.10.4.1.1 CPC No. 4 Front Panel Display Flags .....	29
3.10.4.1.1.1 CPC No. 4 SENSOR# Flag .....	29
3.10.4.1.1.2 CPC No. 4 CLOCKFLAG Flag .....	30
3.10.4.1.1.3 CPC No. 4 ACKNOWLEDGE/STATUS Flag .....	30
3.10.4.1.2 CPC No. 4 Front Panel Switch Handler .....	30
3.10.4.1.2.1 CPC No. 4 Status Clear Switch Function .....	30
3.10.4.1.2.2 CPC No. 4 Sensor/Mode Select Switches .....	31
3.10.4.1.2.3 CPC No. 4 Clock Control Switches .....	31
3.10.4.1.3 CPC No. 4 Front Panel Display Task .....	31
3.10.4.1.3.1 CPC No. 4 Wind Data Display Functions- Indicator .....	32
3.10.4.1.3.1.1 CPC No. 4 Wind Data Display Functions- Recorder .....	32
3.10.4.1.3.2 CPC No. 4 Status Display Function .....	32

3.10.4.1.3.3 CPC No. 4 Clock Setting/Display Function .....	33
3.10.5 Identification of CPC No. 5 .....	33
3.10.5.1 Description of CPC No. 5 .....	33
3.10.6 Identification of CPC No. 6 .....	33
3.10.6.1 Description of CPC No. 6 .....	33
3.10.7 Identification of CPC No. 7 .....	34
3.10.7.1 CPC No. 7 Description .....	34
3.10.7.1.1 CPC No. 7 BIT Error Code Display Functions .....	34
3.10.7.1.1.1 CPC No. 7 Status Display Acknowledge Function .....	34
3.10.7.1.2 CPC No. 7 Description of the CPU Test .....	41
3.10.7.1.3 CPC No. 7 Description of the ROM Test .....	41
3.10.7.1.4 CPC No. 7 Description of the RAM Test .....	41
3.10.7.1.5 CPC No. 7 Description of Inter-Assembly Communication Loop Test .....	41
3.10.7.1.6 CPC No. 7 Description of Inter-Assembly Timeout Error ...	41
3.10.7.1.7 CPC No. 7 Description of Inter-Assembly Multiple CRC Error .....	42
3.10.7.1.8 CPC No. 7 Description of AWDS Loop Test .....	42
3.10.7.1.9 CPC No. 7 Description of Printer Error Routines .....	42
3.10.7.2 CPC No. 7 Interfaces .....	42
3.10.8 Identification of CPC No. 8 .....	43
3.10.8.1 Description of Start Up Modes .....	43
3.10.8.2 Description of CPC No. 8 .....	43
3.11 PROGRAM LISTINGS .....	44
3.11.1 Naming Conventions .....	44
3.11.2 CPC Block Numbers .....	44
4. QUALITY ASSURANCE .....	46
4.1 TEST PLAN/PROCEDURE CROSS REFERENCE INDEX .....	46
4.1.1 Performance Test Simulation Test Concept .....	46
5. PREPARATION FOR DELIVERY .....	49

## LIST OF TABLES

TABLE I	ALLOCATION OF FUNCTION MATRIX .....	4
TABLE II	DISPLAY MEMORY MAP .....	7
TABLE III	FRONT PANEL INTERFACE SWITCH VALUES .....	8
TABLE IV	MCS STATUS BITS .....	8
TABLE V	MCS CONTROL BITS .....	10
TABLE VI	REAL-TIME CLOCK ADDRESSES AND FUNCTIONS .....	10
TABLE VII	INTER-ASSEMBLY OPCODE BIT .....	12
TABLE VIII	INTER-ASSEMBLY OPCODES .....	12
TABLE IX	AWDS OUTPUT FORMAT .....	15
TABLE X	INTERRUPTS .....	16
TABLE XI	CONSTANTS .....	19
TABLE XII	GENERAL STATUS CODES STATUS DISPLAY-LEFT DIGIT .....	35
TABLE XIII	GENERAL STATUS CODES STATUS DISPLAY-RIGHT DIGIT .....	36
TABLE XIV	STATUS DISPLAY MODE CODES FOR THE INDICATOR/RECORDER STATUS WORD WIND DIRECTION DISPLAY-LEFT DIGIT (CODE GROUP 1) .....	37
TABLE XV	STATUS DISPLAY MODE CODE FOR THE INDICATOR/RECORDER STATUS WORD WIND DIRECTION DISPLAY-RIGHT DIGIT (CODE GROUP 2) .....	37
TABLE XVI	STATUS DISPLAY MODE CODES FOR THE INDICATOR/RECORDER STATUS WORD WIND SPEED DISPLAY-LEFT DIGIT (CODE GROUP 3) .....	38
TABLE XVII	STATUS DISPLAY MODE CODES FOR THE INDICATOR/RECORDER STATUS WORD WIND SPEED DISPLAY-RIGHT DIGIT (CODE GROUP 4) .....	38

TABLE XVIII STATUS DISPLAY MODE CODES FOR THE SENSOR STATUS WORD DIRECTION VARIABILITY LEFT DISPLAY-LEFT DIGIT (CODE GROUP 5) .....	39
TABLE XIX STATUS DISPLAY MODE CODES FOR THE SENSOR STATUS WORD DIRECTION VARIABILITY LEFT DISPLAY-RIGHT DIGIT (CODE GROUP 6) .....	39
TABLE XX STATUS DISPLAY MODE CODES FOR THE SENSOR STATUS WORD DIRECTION VARIABILITY RIGHT DISPLAY-LEFT DIGIT (CODE GROUP 7) .....	40
TABLE XXI STATUS DISPLAY MODE CODES FOR THE SENSOR STATUS WORD DIRECTION VARIABILITY RIGHT DISPLAY-RIGHT DIGIT (CODE GROUP 8) .....	40
TABLE XXII STATUS DISPLAY MODE CODES FOR THE SENSOR STATUS WORD GUST SPREAD DISPLAY-RIGHT DIGIT (CODE GROUP 9) .....	41
TABLE XXIII DISTRIBUTED BUILT IN TESTS .....	44
TABLE XXIV CPC LISTING BY BLOCK NUMBER .....	47
TABLE XXV FUNCTION VS TEST CROSS INDEX .....	50



## 1. SCOPE

This specification establishes the requirements for complete identification of the Digital Wind Measuring System Indicator/Recorder Application Program (CPIN 83M-FMQ13-F001-00A) to be formally accepted by the procuring activity. This CPCI is the operation program for the ID-2408/FMQ-13(V) Digital Display Indicator and the RO-588/FMQ-13(V) Wind Direction and Speed Recorder, referred to hereafter as the indicator and recorder, respectively. This documentation was prepared in accordance with Data Item Description DI-E-30145 and MIL-STD-483. The program was developed in accordance with MIL-STD-1679 to the extent specified in paragraph 3.3.11 of Engineering Performance Specification MMA-81-025B.

## 2. APPLICABLE DOCUMENTS

### Specifications

#### Other specifications

MMA 81-025B	22 Apr 85	Engineering Performance Specification for a Wind Measuring System
-------------	-----------	---

### STANDARDS

#### Federal

FED-STD-1010	11 Aug 77	Telecommunications: Bit Sequencing of American National Standard Code for Information Interchange (ASCII) in Serial by Bit Data Transmission
--------------	-----------	--

FED-STD-1011	11 Aug 77	Telecommunications: Character Structure and Character Parity Sense for Serial by Bit Data Communication in the American National Standard Code for Information Interchange (ASCII)
--------------	-----------	--

FED-STD-1020	24 Sep 75	Telecommunications: Electrical Characteristics of Balanced Voltage Digital Interface Circuits
--------------	-----------	---

FED-STD-1030A	31 Jan 80	Telecommunication: Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits
---------------	-----------	--

#### Military

MIL-STD-129H Thru Notice 2	3 Jan 78 1 Jul 80	Marking for Shipment and Storage
-------------------------------	----------------------	----------------------------------

MIL-STD-130F	21 Apr 82	Identification Marking of US Military Property
--------------	-----------	--

MIL-STD-143B	12 Nov 69	Standards and Specifications, Order of Preference for the Selection of
MIL-STD-188C Notice 2	24 Nov 69 17 Nov 76	Military Communication System Thru Technical Standards
MIL-STD-188-114	24 Mar 76	Electrical Characteristics of Digital Interface Circuits
MIL-STD-188-120	15 May 76	Military Communications System Standards and Definitions
MIL-STD-470	21 Mar 66	Maintainability Program Requirements
MIL-STD-483	21 Mar 79	Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs
MIL-STD-471A	27 Mar 73	Maintainability Verification/Demonstration/Evaluation
MIL-STD-1472C	10 May 84	Human Engineering Design Criteria for Military Systems, Equipment & Facilities
MIL-C-1679	1 Dec 78	Weapon System Software Development
Data Item Description		
DI-E-30145	9 May 76	Computer Software/Computer Program/Computer Data Base Configuration Item(s)

### 3. REQUIREMENTS

#### 3.0 GENERAL DESCRIPTION

The Indicator/Recorder Application Program provides sequencing, control, and data processing capabilities to: 1) sample wind speed and direction values, 2) process the wind data to obtain desired direction, speed, and gust information, and 3) output the results on LED displays, or a printer, or over a serial interface or any combination of output channels. The program also has a number of built-in tests (BITs) to assure reliable operation. An overview of the system is provided below to describe the functional relationship between the assemblies which comprise the system. A block diagram is shown in Figure 1.

##### 3.0.1 System Assemblies

The wind measuring system is comprised of three assemblies: 1) sensor, 2) indicator, and 3) recorder. A system can be comprised of these assemblies in various combinations which are specified in MMA 81-25B, paragraph 3.1.1. The sensor measures the direction and speed of the wind and provides these measurements as 5-second averages. (The sensor is a solid state wind measuring device and is controlled by its own microprocessor. The CPCI for the sensor is described in a separate specification.) The indicator displays various wind parameters which the application program computes from the sensor data. The recorder displays and prints wind parameters computed from the sensor data.

### 3.0.2 Inter-Assembly Communications

All the assemblies communicate with each other via an inter-assembly communications network consisting of cabling between the separate units over which serial digital information is transmitted. This network forms a broadcast bus which means information from one assembly will broadcast to all other assemblies simultaneously over the same physical link. The protocol, which governs access to the network, incorporates a master control unit. This master unit, which is switch selectable and can be either an indicator or recorder, polls the sensors for the latest data every 5 seconds. When a sensor is polled it transmits the latest 5-second wind sample three (3) times. This will give each receiving assembly three (3) chances to acquire the data. Each assembly processes the data as soon as it is received. If the master unit does not receive a response or receives a corrupted response it will repoll the sensor.

### 3.0.3 Programming Language

The Indicator/Recorder Application Program is written in the programming language known as FORTH. This particular form of FORTH is called polyFORTH. One of the attributes of polyFORTH is its multi-tasking capabilities. Multi-tasking provides a means of supporting several processing tasks concurrently. This is accomplished by using a round-robin control structure which allows each programming task an opportunity to process data as the need arises. If a task does not have any current need to process data, it is by-passed as the round-robin control progresses. Multi-tasking improves the processing efficiency of the system.

## 3.1 CPCI STRUCTURAL DESCRIPTION

There are eight (8) major components of the Indicator/Recorder Application Program. These components are:

- 1) The Scheduler (which provides fixed timing functions);
- 2) Inter-Assembly Communications (IAC);
- 3) Wind Data Processing (WDP);
- 4) Front Panel Interface (FPI);
- 5) AWDS Interface (AWDS);
- 6) Printer Interface (PI);
- 7) Built In Test (BIT) Routines; and
- 8) Start-Up and Recovery Routines (STUR).

An allocation of function matrix is provided by Table I.

TABLE I  
ALLOCATION OF FUNCTION MATRIX

	<u>Scheduler</u>	<u>Inter-Assembly Comm Task</u>	<u>Process Task</u>	<u>Front Panel Display Task</u>	<u>AWDS Task</u>	<u>Printer Task</u>	<u>BIT Task</u>	<u>Startup Recovery</u>
Sequence Control	*							
I/O Control		*		*	*	*		
Data Processing			*					
Displays				*				
Operational Control				*				
Error Detection		*	*		*	*	*	*
Real Time Diagnostics		*		*	*		*	*
Fault Recovery								*

### 3.2 FUNCTIONAL FLOW DIAGRAMS/CHARTS

Functional diagrams of control and data flow are provided in Figures 2 and 3 respectively.

#### 3.2.1 Control Functional Flow Diagram

Many of the program functions are implemented at fixed time intervals. Sequencing control is provided by the Scheduler which set flags that enable other processes at regular fixed intervals. The Front Panel Interface routine is enabled every 0.5 seconds to refresh the displays and support 1-Hz flash rates. Every 5 seconds AWDS output and Inter-Assembly Communication processes are commenced. Every 60 seconds Printer Interface output is performed on recorder units. The Wind Data Processing routine runs asynchronously from the Scheduler and is driven by the Inter-Assembly routine. Asynchronous interrupt inputs are supported by the Front Panel Interface to accept command inputs via the front panel switches. The Inter-Assembly Communication routine can support interrupts from the network input/output (I/O) port. Figure 2 provides a diagram of the control relations between the various CPCs.

### 3.2.2 Data Flow Diagram

Input data from all assemblies are received into an I/O buffer and processed by IAC. If the input is wind data, it is passed to the Wind Data Input buffer for the particular sensor that produced the input. WDP performs the wind calculations using the new input along with the stored wind parameters of the past 10 minutes. During processing WDP updates the Wind History queues and places calculated output results in the temporary results buffer. Upon completion of processing, all wind data output values are moved from the temporary buffer to the Result buffer for the sensor response just processed. (Each sensor response is processed individually as soon as it is received.) The output routines (Front Panel Displays, AWDS, and printer) move the contents of the Result buffers to their respective output buffer at the start of execution. The AWDS routine copies a buffer for each sensor that is configured into the system. The Printer and Front Panel routines copy the buffer of the active sensor. If a momentary sensor select function is initiated, the Front Panel routine will copy and display the contents of the select sensor's Result buffer. A diagram depicting data flow is given in Figure 3.

### 3.3 INTERFACES

The following paragraphs describe the interfaces of the microprocessor system of the Indicator/Recorder. This includes the protocol used for interprocessor communication, hereafter referred to as inter-assembly communications.

#### 3.3.1 The Indicator/Recorder Microprocessor System

The Indicator/Recorder Assembly is comprised of an 8088 microprocessor with 8 Kbytes of RAM, 16 Kbytes of EPROM, a real time clock (RTC) and five input/output (I/O) interfaces; (1) Inter-Assembly Communication, (2) Automated Weather Distribution System (AWDS), (3) Front Panel Display, (4) Miscellaneous Control and Status (MCS), (5) Printer Data Output (PDO). An 8274 Multi-Protocol Serial Controller (MPSC) dual Universal Synchronous Asynchronous Receiver Transmitter (USART) is used to provide two serial I/O interfaces, one for inter-assembly communications and the other for the AWDS interface. Both serial interfaces are set for 1200 baud. The front panel displays and switches are interfaced by the 8279 Programmable Keyboard/Display Interface Circuit. The parallel interface to the printer is provided by a dedicated design using data latch devices. A programming model of the parallel port is provided in Section 6 along with data sheets for all the interface devices mentioned above. Figures 4 and 5 show the memory map and I/O device register map, respectively.

##### 3.3.1.1 Inter-Assembly Communication Interface

The two channels of the 8274 USART are designated as channel A and Channel B. Channel A is used for inter-assembly communications. The USART is located in the I/O memory at location 0000. Both channels have separate control/status and data registers. The channel A control/status is located at 02 and is referenced by the constant `comm/esr`. The data register is referenced by the constant `comm/data` and is located at 00 in the I/O memory. Channel A is initiated by the routine `comm.init`

(block 610) which set the channel to use a transmit/receive clock of 16 times the baud rate, 8 data bits per character, no parity, and one stop bit. The interrupt functions are initialized with external event, transmitter, receiver, and special receive condition interrupts enabled. Interrupt control is enabled for both channels with 8086/88 mode interrupt acknowledge sequencing for variable vector interrupts. Use of the 8279 USART as the interrupt controller is described in section 3.4.

### 3.3.1.2 AWDS Interface

Channel B is used for the AWDS interface. The channel B control/status is located at 03 and is referenced by the constant `awds/csr`. The data register is referenced by the constant `awds/data` and is located at 01 in the I/O memory. Channel B is initiated by the routine `awds.init` (block 612) which set the channel to use a transmit/receive clock of 16 times the baud rate, 7 data bits per character, even parity, and one stop bit. The interrupt functions are initialized with external event, transmitter, receiver, and special receive condition interrupts enabled. The vector offset is set on the B channel for both channels and is set to vector #8 (address 20(H) in RAM). The carrier detect and clear-to-send functions are enabled and the request-to-send line is asserted.

### 3.3.1.3 Front Panel Interface

The interface for the front panel switches and displays is provided by the 8279 Programmable Keyboard/Display Interface integrated circuit (IC). This IC provides all data storage, timing and sequencing control, interrupt and status information required to drive the displays and sense switch closure status. The control register is located at A1(H) in the I/O memory and is referenced by the constant `fpi/csr`. The data register is located at A0(H) and is referenced by the constant `fpi/data`.

Display storage consist of 16 bytes of RAM (32 4-bit nibbles) which can be accessed as bytes or nibbles. Two sets of four output lines drive the seven-segment drivers for the 32 LED displays on the front panel. These two sets of output lines are designated as the "A" output side (A0-A3) and the "B" output side (B0-B3). The the display memory that drive these outputs are divided into A and B sides. Table II provides a list mapping the display RAM locations of the 8279 to the individual LED displays as designated on the schematic of the display PCBA (DWG # 8121-1017). Table II also provides display names.

The 8279 switch interface consists of 8 input lines called return lines (RL0-RL7) connected to an internal switch debounce circuit and an 8 byte internal RAM section. These eight return lines can be used in conjunction with three scan lines (SL0-SL2) and a 3-to-8 decoder to form an 8 X 8 switch matrix. A subset of only 15 switch inputs are used for this application. Five of of these switch inputs are used to sense configuration jumper settings for the number of sensors, configuration mode switch , and indicator/recorder jumper. The remaining ten inputs handle the front panel switches. One addition switch input is gained with the LAMP TEST switch connected to the control (CNTL) keyboard input. The 8279 is initialized to allow switch closures to generate and IRQ interrupt and are latched into the internal RAM as a number ranging from 0 to 64 [0-40(H)]. Table III provides the hexadecimal value corresponding to the switch/jumper input name.

TABLE II  
DISPLAY MEMORY MAP

<u>MEMORY LOCATION</u>	<u>LED #</u>	<u>DISPLAY NAME</u>
A0	3	DIRECTION - UNITS
A1	2	DIRECTION - TENS
A2	1	DIRECTION - HUNDREDS
A3	6	SPEED - UNITS
A4	5	SPEED - TENS
A5	4	SPEED - HUNDREDS
A6	12	DIR. VAR. 1 - UNITS
A7	11	DIR. VAR. 1 - TEN
A8	10	DIR. VAR. 1 - HUNDREDS
A9	15	DIR. VAR. 2 - UNITS
A10	14	DIR. VAR. 2 - TENS
A11	13	DIR. VAR. 2 - HUNDREDS
A12	9	GUST - UNITS
A13	8	GUST - TENS
A14	7	GUST - HUNDREDS
A15	32	ACTIVE SENSOR
B0	19	HOURS - UNITS
B1	18	HOURS - TENS
B2	21	MINUTES - UNITS
B3	20	MINUTES - TENS
B4	23	SECONDS - UNITS
B5	22	SECONDS - TENS
B6	25	MOUTH - UNITS
B7	24	MOUTH - TENS
B8	27	DAY - UNITS
B9	26	DAY - TENS
B10	29	YEAR - UNITS
B11	28	YEAR - TENS
B12	17	GUST SPREAD - UNITS
B13	16	GUST SPREAD - TENS
B14	31	STATUS - UNITS
B15	32	STATUS - TENS

TABLE III  
FRONT PANEL INTERFACE SWITCH VALUES

HEX VALUE	SWITCH NAME (IND;REC)
0	SENSOR # 1; NORMAN
1	SENSOR # 2; PW-10
2	SENSOR # 3; PW-60
3	SENSOR # 4; PW-24
4	ACTIVE SENSOR; ALARM RESET
5	IND/REC
6	NSNR0
7	NSNR1
8	RUN/SET
10	FIELD SELECT
18	COUNT UP
20	COUNT DOWN
28	STATUS CLEAR
30	MBR0
38	MBR1

TABLE IV  
MCS STATUS BITS

BIT#	SIGNAL NAME	FUNCTION
0	ACK*	ACKNOWLEDGE RECEIPT OF DATA
1	BUSY	PRINTER BUSY
2	PAPMT	PAPER OUT
3	SLCT	PRINTER ONLINE
4	FAULT*	PRINTER FAULT OR ERROR
5	--	--
6	--	--
7	2/10*	2 OR 10 MINUTE AVERAGING

The configuration input labeled "IND/REC" corresponds to the "E@" jumper on the interconnect PCBA (DWG# 8121-1029) establishes the operation configuration as an indicator if the jumper is open, or as a recorder if the jumper is installed. The configuration inputs labeled "NSNR0" and "NSNR1" correspond to jumpers E3 and E4 on the interconnect PCBA and establishes for the software the number of wind sensors in the system. These jumpers form a two bit value ranging from 0 to 3 corresponding to 1 to 4 sensors, respectively. The configuration inputs "MBR0" and "MBR1" correspond to the inputs from the CONFIGURATION SWITCH located on the back of the indicator/recorder assembly. This switch is connected to E20, E21, and E22 on the interconnect PCBA and forms a two bit value from 0 to 2 corresponding to BACKUP, REGULAR, and MASTER modes, respectively. These modes invoke different communication functions. The MASTER mode allows a unit to poll sensors for data. The REGULAR mode allows a unit only to listen for communications. The BACKUP mode allows a unit to start polling if the MASTER unit fails to poll within a set amount of time. This protocol is explained in detail in the following section.

#### 3.3.1.4 Real Time Clock(RTC) Interface

The RTC interface is provided by the 7170 Real-Time Clock I.C. The RTC is used to provide regular interrupts at 0.1 second, one second and one minute intervals as well as maintaining the time and date for the front panel display. It has a base address of 0020(H) in I/O memory. Table IV provides the addresses, functions, and program constant names used to reference the clock. The clock has an internal RAM used to store the time and date in a battery backup mode in the eventuality of a power loss.

#### 3.3.1.5 Miscellaneous Control and Status (MCS) Register

The MCS register is an 8 bit register made from MSI logic devices comprised of U4 and U6 on the Digital Display PCBA (DWG# 8121-1017). This register is located in I/O memory at E0(H) and is referenced by the constant mcs/reg. Reading this register provides the status bits from the printer and one the state of "2/10" configuration jumper. This jumper corresponds to E1 on the interconnect PCBA. It is used to establish the averaging period for the wind direction and wind speed routine on the recorder only. Table V defines the status bits for this register. Writing to this register controls the printer interface lines, the alarm buzzer on the recorder, and the four discrete LEDs on the front panel. Table VI defines the control bits of the MCS register. The printer interface is J3 on the interconnect PCBA.

#### 3.3.1.6 Printer Data Output Interface

The type of printer interface is known as a Centronix 8-bit parallel interface. It is comprised of U5 on the Digital Display PCBA (DWG # 8121-1017) and is located at C0(H) in I/O memory. Output data to the printer is written to this location. Status and control of the printer is accomplished by the MCS register described above.

#### 3.3.1.7 Watchdog Timer Interface

The microprocessor PCBA has a watchdog timer circuit. This timer will reset the microprocessor if it is allowed to time out. The program resets this timer as part of the Built-In-Test routines. If something causes the the program to stop proper execution the watchdog timer will not get reset and will time out in approximately 0.5 seconds and reset the microprocessor. The timer is reset by writing to location 40(H). The timer is referenced by the constant watchdog.

TABLE V  
MCS CONTROL BITS

BIT#	SIGNAL NAME	FUNCTION
0	STRB*	DATA INPUT STROBE
1	PRM*	PRINTER RESET LINE
2	--	(NOT ASSIGNED)
3	ALRM*	ALARM
4	LS1	LED 1
5	LS2	LED 2
6	LS3	LED 3
7	LS4	LED 4

TABLE VI  
REAL-TIME CLOCK ADDRESSES AND FUNCTIONS

LOCATION	FUNCTION	CONSTANT	
20	COUNTER 1/100 SECONDS	clock	
21	COUNTER-HOURS	HOUR	
22	COUNTER-MINUTES	MINUTE	
23	COUNTER-SECONDS	SEC	
24	COUNTER-MONTH	MONTH	
25	COUNTER-DATE	DAY	
26	COUNTER-YEAR	YEAR	
27	COUNTER-DAY OF WEEK	--	
28	RAM- 1/100 SECONDS	--	
29	RAM-HOURS	RAMHOUR	
2A	RAM-MINUTES	RAMMIN	
2B	RAM-SECONDS	RAMSEC	
2C	RAM-MONTH	--	
2D	RAM-DATE	--	
2E	RAM-YEAR	--	
2F	RAM-DAY OF WEEK	--	
30	INTERRUPT STATUS AND MASK		clock/intr
31	COMMAND REGISTER	clock/csr	

### 3.3.2 Inter-Assembly Communication Protocol

Inter-assembly communication is accomplished using a shared multi-drop serial data link configuration. Data transmission is accomplished using Bell 202 compatible FSK techniques. Asynchronous serial data transmissions occur at a 1200 baud rate over a physical network comprised of a single pair of wires (2 wires) connected to each assembly. Each character frame consists of 1 start bit, 1 stop bit, and 8 data bits with no parity generation. This signal format is used to transmit messages between the assemblies. The general protocol uses a central control element to govern the use of the network. This controller is referred to as the master assembly. The master assembly has the responsibility of polling the sensors every five (5) seconds. There is only one MASTER assembly allowed in a system. A polling session starts every 5 seconds and commences with a poll request to sensor ID#-1. (The sensor ID#s are established by jumper setting in the sensor assembly. Sensor ID#s must be assigned starting with ID#-1 and running consecutively up to a maximum of four for inter-assembly communications to work properly.) When sensor #1 receives the poll request it transmits the latest 5-second average X and Y wind values three times. This poll response and all inter-assembly transmissions are broadcast over the network to all assemblies to receive. After the master receives the poll response from sensor #1 it polls sensor #2. This process continues until each sensor in the system is polled. The "number of sensor" (NSNR) jumpers on the indicator/recorder assembly establish how many sensors will be polled by the MASTER. There are several negative acknowledge and redundancy schemes used in the protocol which is described in detail in paragraph 3.10.2.

#### 3.3.2.1 Inter-Assembly Communication Message Format

The protocol uses message packets to exchange information between assemblies. Each message is framed by control character pairs (DLE STX and DLE ETX) and has a 16-bit cyclic redundancy check (CRC) number affixed to the end of the message for error checking as shown here:

DLE, STX, MESSAGE, DLE, ETX, CRC, CRC

##### 3.3.2.1.1 Inter-Assembly Communication Opcode

Each message contains an Opcode as well as data. The Opcode character has eight bits of which only four are used for the actual operation code. The other four bits are used for identification purposes. Two bits are used to indicate the active sensor number and two bits are used for the sensor ID number. These bit assignments are provided in Table VII. The format of the Opcode character and a list of the codes are provided in Table VIII.

##### 3.3.2.1.2 Inter-Assembly Communication Message Data

The information contained in the data field of a poll request sent by the master assembly is a modulo 16 poll count used by all the assemblies to track the sequence of poll requests and poll responses between the master and the sensors. Use of the poll count is described further in paragraph 3.10.2.2.4. A sensor response data field consists of wind data, sensor status, and an echo of the poll count received with the poll request from the master assembly. Normally, four bytes of wind data are transmitted. However, if the data value

TABLE VII  
INTER-ASSEMBLY OPCODE BIT

BIT#	DESCRIPTION
0	OPCODE BIT 0
1	OPCODE BIT 1
2	OPCODE BIT 2
3	BACK UP OPCODE BIT
4	SENSOR ID BIT 0
5	SENSOR ID BIT 1
6	ACTIVE SENSOR ID BIT 0
7	ACTIVE SENSOR ID BIT 1

TABLE VIII  
INTER-ASSEMBLY OPCODES

- 0 - Illegal Code
- 1 - Poll
- 2 - Repoll
- 3 - Poll Response
- 4 - Repoll Response
- 5 - (unassigned)
- 6 - (unassigned)
- 7 - (unassigned)
- 8 - (unassigned)
- 9 - Backup Poll
- A - Backup Repoll *Response*
- B - Backup Repoll
- C - Backup Repoll Response
- D - (unassigned)
- E - (unassigned)
- F - Master Take Over

is the same as a DLE, then a second DLE character is sent to indicate that the first DLE was a data value and not a control character (e.g. DLE, ETX or DLE, STX). This provides a "transparency" of a DLE value occurring in the data field. In worst case this could produce eight bytes of wind data values.

### 3.3.2.1.3 Inter-Assembly Communication Message Timing

Communications are carried out using 1200 baud as the transmit and receive rate. This works out to be 8.33 milliseconds/character. Using this rate a poll request takes 66.64 milliseconds to send the following eight bytes of information:

DLE, STX, OPCODE, POLLCNT, DLE, ETX, CRC, CRC

The data message in the poll request is the poll count (POLLCNT) which is a modulo 16 count used to track message sequencing.

A poll response from the sensor will nominally require 108.3 milliseconds to send the following message :

DLE, STX, OPCODE, POLLCNT, <sup>2</sup>WD, <sup>2</sup>WD, <sup>2</sup>WD, <sup>2</sup>WD, STATUS, <sup>4 to 10's</sup>DLE, ETX, CRC, CRC

WD indicates one byte of wind data. The worst case time which corresponds to eight bytes of wind data (four extra DLE characters) is equal to 150 milliseconds. Worst Case Poll Response:

Each wind data field requires a DLE to follow

DLE, ETX, OPCODE, POLLCNT, DATA1...DATA8, STAT1, STAT2, DLE, STX, CRC, CRC

18 Bytes (8.33 ms/byte)  $\approx$  150 ms

A timing diagram of each component of a poll request and poll response as well as a complete poll session for four sensors is shown in figure 51. This timing diagram uses the worst case time of 150 ms for the poll response and the maximum latency periods for the carrier detect off-to-on and on-to-off transition.

### 3.3.3 AWDS Interface Message and Protocol

The AWDS port is used to interface to an automated data distribution network. The interface is configured as an unbalanced generator-receiver per MIL-STD-188-114 with provisions for optional wiring for the grounding and balancing arrangements provided. The interface is configured as a Send Only type which mandates that Signal Ground (SG), Send Data (SD) Send Common (SC), Receive Common (RC), Data Mode (DM), Request to Send (RS), Clear to Send (CS) and Test Mode (TM) lines shall be present. The RS line is always asserted. Data can be transmitted only if CS and DM are also asserted. Transmission of messages begins every five seconds with one message per sensor. The message is transmitted in ASCII characters. A list of the information contained in each field of the message, the length of the field and the field position within the message is provided in Table IX. Figure 52 provides the format of a sample message.

### 3.4 PROGRAM INTERRUPTS

The following paragraphs describe the program interrupts: specifically, the source, purpose, type, priority, and required response for each interrupt. The presentation will progress from highest to lowest priority. All interrupts are generated by the 8274 USART which is used as an interrupt controller for the real-time clock and the front panel interface as well as the communication functions of port A and port B. Table X provides a list of the interrupt vector number, the RAM location of the vector, interrupt handler program name, the source of the interrupt and the function of the interrupt.

#### 3.4.1 Inter-Assembly Receiver Interrupt

This interrupt is generated when a character is received via the inter-assembly communication interface. This is an IRQ interrupt for the purpose of processing received characters and has the highest priority level. This interrupt is serviced by the Inter-Assembly Interrupt Handler described in Paragraph 3.10.2.3. The interrupt rate will be once every 8.33 milliseconds during receipt of messages and will nominally have a 30% duty cycle for a four-sensor system.

#### 3.4.2 Inter-Assembly Communication Transmitter Interrupt

An IRQ type interrupt occurs each time a character is transmitted when the transmit buffer becomes empty. This is serviced by the Inter-Assembly Output Handler which transmits another character if one is available. The interrupt is second in priority and will occur at a rate of one each 8.3 milliseconds during poll request transmission periods. This would occur only in master or backup assemblies. The frequency of the interrupt will be 120 pps. The duty cycle will range from a low of 1.5% for a one sensor system to a high of 6% for a four sensor system.

#### 3.4.3 AWDS Receiver Interrupt

This interrupt is generated when a character is received via the AWDS interface. It is enabled only during the AWDS built-in-test which occurs at the beginning of each AWDS output message. This is an IRQ interrupt and is third in priority. The service routine is the AWDS Receiver Interrupt Handler which checks the validity of the received character. The highest interrupt rate would be approximately once each second for a four-sensor system.

#### 3.4.4 AWDS Transmitter Interrupt

An IRQ type interrupt occurs each time a character is transmitted when the transmit buffer becomes empty. This is serviced by the AWDS Output Handler which transmits another character if one is available. The interrupt is

TABLE IX  
AWDS OUTPUT FORMAT

FIELD NAME	CONTENT	FIELD LENGTH	FIELD POSITION
System Nomenclature	FMQ13V	6	1
Sensor Number	01 - 04	2	7
Message Count (Modulo 100)	00 - 99	2	9
Valid Data Flag	? or 0	1	11
Character Count	090	3 *	12
Meteorological Data			
Date	MM/DD/YY	8 *	16
Time	HH:MM	5 *	25
Wind Direction	000-360	3 *	31
Wind Speed	000-250	3 *	35
Gust	000-250	3 *	39
Direction Variability 1	000-360	3 *	43
Direction Variability 2	000-360	3 *	47
Gust Spread	00-99	3 *	51
10 Min. Peak Wind Direction	000-360	3 *	55
10 Min. Peak Speed	000-250	3 *	59
10 Min. Peak Time	HH:MM	5 *	63
60 Min. Peak Wind Direction	000-360	3 *	69
60 Min. Peak Speed	000-250	3 *	73
60 Min. Peak Time	HH:MM	5 *	77
24 Hour Peak Wind Direction	000-360	3 *	83
24 Hour Peak Speed	000-250	3 *	87
24 Hour Peak Time	HH:MM	5 *	91
Std-Dev. of Direction	000-180	3 *	97
Active Flag	N or A	1 *	101
Status	00 - 77	2	103
Message Termination	CR - LF	2	105

\* - Field Followed by one Space Character

TABLE X  
INTERRUPTS

VENDOR#	LOCATION	HANDLER NAME	SOURCE	FUNCTION
0	0-3 (H)	DIVERR	INTERNAL-DIVISION BY 0	
8	20-23 (H)	INTR0	PORT B TRANSMIT INTERRUPT	AWDS TRANSMITTER INTERRUPT
9	24-27 (H)	INTR1	PORT B EXTERNAL STATUS INTER	RTC, FRONT PANEL INTERRUPT
10	28-2B (H)	INTR2	PORT B RECEIVE DATA INTERRUPT	AWDS RECEIVE INTERRUPT
11	2C-2F (H)	INTR3	PORT B SPECIAL RECEIVE INTER	PARITY ERROR, OVERRUN, FRAMING
12	30-33 (H)	INTR4	PORT A TRANSMIT INTERRUPT	INTER ASSY TRANSMITTER INTERRUPT
13	34-37 (H)	INTR5	PORT A EXTERNAL STATUS INTER	CARRIER DETECT INTERRUPT
14	38-313 (H)	INTR6	PORT A RECEIVE DATA INTERRUPT	INTER ASSY RECEIVER INTERRUPT
15	36-3F (H)	INTR7	PORT A SPECIAL RECEIVE INTER	PARITY ERROR, OVERRUN, FRAMING



fourth in priority and will occur at a rate of one each 8.3 milliseconds during data transmission periods which starts once every five seconds. A single message contains 107 ASCII characters. There is one message sent for each sensor configured in the system. The frequency of this interrupt will be 120 pps. The duty cycle will range from 18% to 72% depending on the number of sensors configured in the system.

#### 3.4.5 Inter-Assembly External Status Interrupt

This interrupt can be generated by three events: (1) a change of the state of carrier detect, (2) a real-time clock interrupt, or (3) front panel switch interrupt. These events are serviced by the Inter-Assembly Interrupt Handler, the Scheduler, and the Front Panel Switch Interface Handler, respectively. A change in carrier status is used to detect the end of the receipt of a message. The real-time clock is used to sequence scheduled events and requires the determination of the time interval that caused the interrupt. The Switch Handler interprets the command issued via the front panel switches. The real-time clock interrupts once each 100 milliseconds and is the prevalent interrupt source of the three. Each on these devices sets a different bit in the read register 0 (RR0) of channel A of the 8274 USART; (1) bit 3 for the carrier detect input, (2) bit 4 for the front panel interface interrupt input, (3) bit 5 for the real-time clock interrupt input.

#### 3.4.6 AWDS External/Status Interrupt

This interrupt is generated when there is a change in the DM, CS, or TM control lines of the AWDS RS-449 interface. The status of these lines controls the AWDS output routine. This is an IRQ interrupt and has the lowest priority. The service routine records the state of the interface line for use by the AWDS output task. Frequency of interrupt is expected to be very low.

#### 3.4.7 Special Receive Condition Interrupts

These interrupts are generated by the 8274 USART if a parity error or framing error or receiver overrun condition is detected. The service routines for these interrupts reads the register and resets the interrupt flag. No other action is taken on these conditions. The frequency of these interrupts is unknown.

### 3.5 TIMING AND SEQUENCING DESCRIPTION

Figure 2 shows the timing and sequencing relations between the CPCs. The Real-Time Clock (RTC) interrupt is the basis for program timing. The Scheduler services the interrupts of the RTC and sets flags to enable various tasks. The displays are refreshed every 0.5 seconds. Every 5 seconds the Inter-Assembly Communication Task is enabled along with the AWDS Task. On recorders, the printer task is enabled every 60 seconds. The Watchdog Timer must be reset within 0.5 seconds or it will time out, causing a reset and startup. Wind data processing is enabled by the Inter-Assembly task after it has processed any received data.

### 3.6 SPECIAL CONTROL FEATURES

There are no special control features involved with the operation of this CPCI.

## 3.7 STORAGE ALLOCATION

The storage allocation for the Indicator/Recorder assembly is confined to the microprocessor solid state memory devices (i.e. EPROM and RAM). Figure 6 shows the memory map of the microprocessor. The RAM is comprised of an 8K x 8 bit static memory device.

### 3.7.1 Data Base Definition

Figure 6 gives a graphical representation of the database for the Indicator/Recorder assembly. The following paragraphs provide a detailed description of the database. The entire database is contained in RAM.

#### 3.7.1.1 File Description

Each sensor has a file section which contains wind data. This file is comprised of 5-second samples of wind data, 24-hour peak wind data, results of the latest calculation period, and values used to provide calculated results. These data items are described in the following paragraphs.

#### 3.7.1.2 Table Description

Much of the information used to produce the calculated wind values is stored in data queues. A data queue is simply a list of the values organized in a sequence from oldest to newest value. There are queues for speed data, direction data, and gust data, all stored as 120 sixteen bit values. There is also a queue for the latest 24 hourly peak wind data; this queue is used to determine the 24-hour peak wind value of speed, direction, and time of occurrence. The remaining data structures are buffers of data items described in the following paragraphs.

#### 3.7.1.3 Item Description

The Results buffer contains the most recent calculated values of wind data. This includes: wind direction to the nearest degree, wind speed to the nearest knot, gusts to the nearest knot, direction variability to the nearest degree, gust spread to the nearest knot, peak wind direction (10 minute, 60 minute, and 24 hour) to the nearest degree, peak wind speed (10 minute, 60 minute, and 24 hour) to the nearest knot, peak wind time of occurrence to nearest minute, and standard deviation of wind direction to the nearest degree. A scratch pad buffer is used to maintain support data such as running sums of orthogonal values over two- and ten-minute periods.

#### 3.7.1.4 Graphic Table Description

Figure 6 provides a graphical representation of the tables within the database.

#### 3.7.1.5 CPCI Constants

Several constants used to reference device registers were defined in the section cover interfaces. Table XI lists the remaining constants used in this CPCI.

TABLE XI  
CONSTANTS

NAME	VALUE	FUNCTION
1MIN	12	ONE MINUTE OFFSET POINTER INTO DATA QUEUE
2MIN	24	TWO MINUTE OFFSET POINTER
10MIN	120	TEN MINUTE OFFSET POINTER
CELL	2	DEFINES 16 BIT WORD AS CELL
1MINLIMIT	12	ONE MINUTE LIMIT FOR INVALID DATA
2MINLIMIT	24	TWO MINUTE LIMIT
10MINLIMIT	120	TEN MINUTE LIMIT
60MINLIMIT	720	60 MINUTE LIMIT
24HRLIMIT	17280	24 HOUR LIMIT
DIRQ	240	OFFSET TO DIRECTION QUEUE
GUSTQUE	480	OFFSET TO GUST QUEUE
PVDDIR	720	OFFSET TO 10 MINUTE PEAK WIND DIRECTION QUEUE
PVDSPD	768	OFFSET TO 10 MINUTE PEAK WIND SPEED QUEUE
PVDTIM	816	OFFSET TO 10 MINUTE PEAK WIND TIME QUEUE
RESULTBUFFER	864	OFFSET TO RESULT BUFFER
PEAKRESULTBUF	896	OFFSET TO 1 & 24 HOUR PEAK WIND RESULT QUEUE
INBUF	910	OFFSET TO INPUT BUFFER
SCRATCH	920	OFFSET TO SCRATCH PAD AREA
TIMEQUE	960	OFFSET TO TIME STAMP QUEUE



### 3.7.2 CPC Relationship

Figure 3 provides a graphical depiction of the relationship of the data base to each CPC that directly accesses the data base.

### 3.8 OBJECT CODE CREATION

The programs of this CPI are written in the FORTH programming language and are compiled on an IBM PC computer system (or equivalent) with 256 KB of RAM and dual floppy disks. The specific version of FORTH used is 8086/IBM PolyFORTH II by Forth, Inc. The procedures for using this programming language can be found in the "PolyFORTH II Reference Manual" and the "PolyFORTH 8086 CPU Supplement to the polyFORTH II Reference Manual". Program code is created using the "Digital Wind Measuring System Target Compiler" (SPIN 83M-FMQ13-F003-00A) supplied with this CPI. The target compiler disk provides the FORTH operating system with its utilities for disk formatting and copying, printing and editing. During operation the target compiler resides in the "A" floppy disk drive of the PC with the "Digital Indicator/Recorder Application Program" disk in drive "B". Code compilation is commenced by entering the following instructions; "EMPTY COMPILER LOAD IBM LOAD" followed by carriage return (CR). The compiler will list the location of each word compiler to the monitor screen. Compilation will continue for approximately five minutes and will compile word in the order define by the load screen commands defined in block 325 through 328 of the application program listing. The object code will reside in memory at end of this procedure. The following command will download the object code from memory to the COM1 serial output port; "REMOTE SEND HEX 0 C000 4000 RDUMP (CR)" This command will download 16384 bytes (4000 hex) starting at RAM location 0 and transcribing it into a load module starting at address C000(H).

### 3.9 ADAPTATION DATA

There is no required direct data entry for functional adaptation. All adaptation is a function of hardware configuration links and switch settings. Setting of these links is described in chapter 2 of the Maintenance Manual TO 31M1-2FMQ13-2.

### 3.10 DETAIL DESIGN DESCRIPTION

This paragraph contains technical descriptions of the computer program components identified in paragraph 3.1 of this specification. The charts and instruction listings which specify the exact configuration of the "Digital Indicator/Recorder Application Program" are contained in drawing number 8200-1010.

#### 3.10.1 Identification of CPC No. 1

This section describes the method of sequence control and timing used on the Indicator/Recorder assemblies. The routine which performs this function is called the Scheduler and is identified as CPC No. 1. The Scheduler functions are implemented in the real-time clock interrupt service routine contained in the routine labeled "<RTC>" listed in blocks 569 through 574.

##### 3.10.1.1 Description of CPC No. 1

Flow charts of the Scheduler routine are shown in Figures 7 through 9. The Scheduler receives interrupts from the real-time clock (RTC) every 100 ms. The interrupt register of the RTC has status bits for different time intervals to indicate the state of the clock at each interrupt.

#### 3.10.1.1.1 Minute Interrupt

The routine reads the register to determine if a minute interrupt has occurred. If so, the minute register is tested to see if it is 55 minutes past the hour, in which case the variable 55FLAG is set to be used by Process (CPC No. 3) to produce the 60-minute and 24-hour peak wind output values. At the 55 minute interrupt a counter is set which is used to delay the the printer output for six seconds to allow processing of the 60 minute and 24 hour peak wind information. Each minute a flag is set which causes a new standard deviation of direction to be produced. Another flag is set on recorder assembly to commence printer output and to update the recorder display values.

#### 3.10.1.1.2 Second Interval Interrupt

The second interval interrupt indicates the passing of a one-second interval. The interrupt is used to indicate 5- and 60-second intervals. The 60-second interval counter is used only if a momentary sensory display function has been initiated. The 60-second interval is the timeout period for this function. The 5-second interval counter is used to enable the AWDS output routine and enable the display update routine on indicator assemblies. The 5-second interval is also used to enable the polling process of the Inter-Assembly Communication Task.

#### 3.10.1.1.3 100 MS Interrupt Interval

The 100-ms interrupt interval is used to measure a 500-ms interval used as the refresh time on the front panel displays. These displays are refreshed every 500-ms. This refresh interval allows a one second flash rate. The 100-ms interval is also used to implement a 200-ms timeout interval to support the sensor polling function of the Inter-Assembly Communication Task (CPC No. 2).

### 3.10.2 Identification of CPC No. 2

The Inter-Assembly Communication CPC is described in this section. The routines that implement the the Inter-Assembly Communication functions are the Inter-Assembly Communication Task listed in blocks 530 through 562 and the inter-assembly communication interrupt service routines "<INTR4>", "<INTR5>", and <INTR6> listed in blocks 593 through 597. This CPC performs control and error checking functions to assure effective and reliable acquisition of wind sensor data by all Indicator/Recorder assemblies. The basis for reliable communication is established by the form and protocol used for data transmission described in Section 3.3.2. This is enhanced by the redundancy and error checking provided by the Inter-Assembly CPC described below.

#### 3.10.2.1 CPC No. 2 Description of Modes of Operation

The Inter-Assembly CPC provides serial I/O data processing, sensor polling control, and message error checking to assure reliable communications. There are three modes of operation which are selected by a switch setting on the back of each assembly. These modes are referred to as Master, Backup, and Normal. The function of each mode is described in detail in Paragraphs 3.10.2.1.1 through 3.10.2.1.3.

To summarize these functions: An assembly in Master mode polls the sensors for data every 5 seconds and processes the responses. An assembly in Normal mode receives all communications and processes wind data. An assembly in Backup mode will function as if in Normal mode unless the master quits polling, in which case the backup assembly will start polling. This arrangement provides redundancy of the system control element.

### 3.10.2.1.1 CPC No. 2 Description of Master Mode

Each system will have one and only one assembly designated to operate in Master mode. In this mode the assembly is the primary sensor polling device. No other assembly can poll the sensors as long as the Master assembly is operating properly. The polling sequence is described in Paragraph 3.10.2.1.1. The only other unique function performed by the Master assembly is its interaction with the designated Backup assembly which is described in Paragraph 3.10.2.1.3. All the other inter-assembly communication functions are a part of Normal mode operation described in Paragraph 3.10.2.1.2.

#### 3.10.2.1.1.1 CPC No. 2 Sensor Polling Sequence

The Master assembly will poll each sensor in the system every 5 seconds. If there is no response from a sensor or if the response is corrupted, then the Master will repoll the sensor at fault. A second repoll will be sent if no satisfactory response is received. If there is still no satisfactory response from the sensor, the Master assembly will indicate an error condition and will proceed with polling the other sensors of the system. A polling cycle is comprised of a polling session of each sensor in the system. A polling session consists of a poll of a single sensor and up to two repolls of the same sensor. If a sensor fails to satisfactorily respond through two consecutive polling cycles, the Master assembly will subsequently poll the faulty sensor only once each cycle until a good response is received. At that point the same repolling strategy will be used again until two consecutive faulty responses are received over two consecutive polling cycles.

#### 3.10.2.1.2 CPC No. 2 Description of Normal Mode

An assembly in Normal mode listens to all inter-assembly communications, tracking the poll/response sequence and extracting wind data and system status information. Each inter-assembly message has a poll count number as described in Paragraph 3.3. This modulo 16 count is generated by the master assembly and is incorporated into the sensor response. Each assembly uses the count to determine if any messages have been missed and how many. The count is also used to determine that a retransmitted message has already been received and processed. If the sequence is broken, an error code will be displayed. If no messages are received over a period of 5 seconds, the assembly will indicate a timeout error code.

#### 3.10.2.1.3 CPC No. 2 Description of Backup Mode

One and only one assembly in each system should be placed in backup mode. This assembly will function as a normal assembly until two consecutive 5-second timeout errors occur. At this point, the backup unit will assume the polling responsibilities of the master assembly. The Opcodes used by the backup assembly for polling are different from the Master Opcodes. These codes can be detected by all assemblies, thereby informing them that the backup assembly is operating as master. The detection of the backup Opcode causes an error code to be displayed on all operating assemblies. If the master assembly starts functioning or if another assembly is enabled to be Master, it will listen for inter-assembly communication. Upon detection of the backup Opcode, the master will wait until the current polling cycle is complete, at which time a "master takeover" command will be issued. This informs the backup unit that a master unit is in control and the backup will halt any further polling and will revert to standby operation.

### 3.10.2.2 Detailed Description of the Inter-Assembly Task

The Inter-Assembly Task is diagramed in Figures 10 through 15. Each of the following subparagraphs will describe one figure.

#### 3.10.2.2.1 Description of Inter-Assembly Task Figure 10

The task starts by initializing a counter used for 5-second timeout detection and initializing input buffers to receive data. The routine then enters a wait state which is called a sleep state. This state can be interrupted by the Inter-Assembly I/O handler which services the serial interface, or by the scheduler routine. The Scheduler can set two flags, the poll flag or the timeout flag. The scheduler will set the poll flag only on the master assembly or on a backup assembly which is acting as the system master. If either flag is set, the routine branches to service the separate conditions. If neither flag is set, the Inter-Assembly I/O handler must have interrupted the wait state and the I/O buffer is processed. The data is checked for validity. If valid the input message is checked for a change in the active sensor. If the sensor has been changed, the new sensor number is stored. If the assembly is a recorder, the alarm is activated.

#### 3.10.2.2.2 Description of Inter-Assembly Task Figure 11

Figure 13 describes most of the polling functions. If the poll flag was set by the scheduler, the program will branch to the point labeled 3. The poll flag is cleared and two counters are initialized. The I counter tracks the number of poll sessions while the N counter tracks the number of polls during a session. There is a maximum of three, one poll and two repolls. The poll message is formatted and transmitted. A timeout counter is set for 200 ms which is the no-response timeout period. The input buffer is setup to receive a response and the task again enters a wait state. This state will again be interrupted by the scheduler or by the I/O handler if a response is received. If the timeout flag is set, an error condition has occurred. The I and N counters are tested for limits. If the poll session is incomplete, then a repoll message is sent and the routine loops back to the point labeled B. If the poll session is complete but the responses were invalid, then the poll fail counter is incremented and the bad data flag is set in the Process Task input buffer. If any of the responses are good, then the data is moved to the process input buffer and the fail poll counter is cleared. After the response is handled appropriately, the I poll counter is incremented and tested against the number of sensors in the system. If more sensors should be polled, the routine loops back to the point labeled A. If all sensors have been polled, the routine branches back to the beginning of the program at the point labeled 1 on Figure 10.

#### 3.10.2.2.3 Description of Inter-Assembly Task Figure 12

If the 5-second timeout flag was set by the scheduler, an entire polling cycle has been missed. The routine branches to the code segment diagramed in Figure 14. The first step is to set bad data flags for the same number of wind data files as there are sensors in the system. A timeout failure counter is used to keep track of the number of consecutive failures. The backup assembly tests this counter. If the count equals two (2), then the backup assembly will setup to start polling sensors. The backup assembly computes the new pollcount and sets the control flag which tells the Scheduler to set the poll flag. The routine then loops back to the start of the routine.

#### 3.10.2.2.4 Description of Inter-Assembly Task Figure 13

This code segment is entered from Figure 12 after receipt of valid data. This diagram shows the use of the pollcount number, poll number, and the opcode of the received message to determine that the proper sequence of messages has been received. The poll number is generated by the master assembly. The opcode is generated by the sending assembly, which is either the master or a sensor. The pollcount is maintained by each assembly. The routine starts by testing the opcode to see if the message was a poll request, repoll request, poll response or repoll response. In the case of a poll request or poll response, the poll number should exceed the current pollcount value by 1, if the assembly has received all messages. In the case of a repoll request or response, if the assembly is in phase with the message sequence (i.e. received the original poll request/response) then the poll number will equal the pollcount. If the assembly is in sequence, then the program loops back to the start. If the sequence has been lost, the timeout failure count is multiplied by the number of sensors to establish a new pollcount. If the calculated pollcount equals poll number, then the sequence has been re-established. If the pollcount is not correct, the failure counters for the affected sensors are incremented.

#### 3.10.2.2.5 Description of Inter-Assembly Task Figure 14

This flow chart continues from the flow chart in Figure 13. The program segment starting at connector label 7 shows the procedure to handle a takeover command. The backup assembly will clear the control flag and the timeout failure counter. This will cancel the backup operation. All other assemblies ignore this command. The program segment starting at connector label 8 shows the procedure for the receipt of a valid poll response which is in the proper sequence. The response in the I/O buffer is placed in the process buffer and the flag is set to indicate receipt of good data. The pollcount is incremented. A running master assembly would not normally execute this program segment because it would have received and processed the response to its own poll message. This program segment is executed by assemblies in normal operating mode. If the assembly is switched to master mode, then this new master would wait until the sensor response number equals the number of sensors in the system and would then transmit the master takeover command.

#### 3.10.2.2.6 Description of Inter-Assembly Task Figure 15

This flow chart is continued from Figure 13 and shows the procedure followed if the Pollcount sequence is lost during a poll request or repoll request. The program segment starts with a calculation of the Pollcount using the timeout failure count and the number of sensors in the system. This calculation represents the pollcount plus the number of messages lost. If this calculation agrees with the present Poll number, then the Pollcount is re-established, the Timeout Failure count is cleared and the routine loops back to the start of the task. If the calculated Pollcount is not correct then the Bad Data Flag is set on each Process Buffer for the calculated number of missed messages. This will indicate to the Process Task that data is missing and will cause a recovery process to start, and will flash the displays of the affected wind parameters. The routine then re-establishes the Pollcount, clears the Timeout Failure count and loops to the start of the task.

### 3.10.2.3 Detailed Description of the Inter-Assembly Communication Interrupt Handler

A flow chart of the interrupt handler is provided by Figures 16 and 17. This is the interrupt service routine for the Inter-Assembly serial interface. This routine receives the characters which comprise inter-assembly messages and checks for proper framing of the message (i.e. DLE, STX, DLE, ETX framing). The main branch of this routine tests the UART Status Request (USR) to determine the cause of the interrupt. The action taken depends on whether the cause was a carrier detect change, a received character, or the transmission of a character. Each of these events are described below.

#### 3.10.2.3.1 Interrupt by Carrier Status Change

An interrupt is generated if the state of the carrier signal changes from on to off or off to on. If the source of the interrupt is a change of carriers from on to off, then a test is made to determine if an STX character was received. This would indicate the receipt of at least part of a legitimate message. If the STX flag is set, then the I/O buffer flag is set and the routine then wakes up the Inter-Assembly Communication Task (IACT). The I/O buffer flag tells the IACT that a message has been received. If the STX flag is not set, then the routine wakes up the IACT without setting the I/O buffer flag. The IACT will ignore the buffer contents and setup to receive new data.

#### 3.10.2.3.2 Transmit Buffer Empty Interrupt

The Transmit Buffer Empty (TBRE) interrupt indicates that the UART is ready for a new character. The character counter (CTR) is tested to see if it is greater than zero, which would mean that there are more characters to be transmitted. If CTR is positive, then a new character is transmitted and CTR is decremented. If CTR is not positive, then the UART is tested to see if the transmission of the last character is complete. If transmission is complete, then the carrier is turned off and the IACT is awakened.

#### 3.10.2.3.3 Received Character Interrupt

If a new character is received, the routine resets a 200 ms Timeout counter used by the master assembly to timeout poll responses. The majority of this portion of the routine is used to detect the proper sequence at the start of a message. If the STX flag is clear, then the DLE flag is tested. If this flag is clear, then the character is tested to see if it is a DLE character. If it is, the DLE flag must be set or the routine will return from the interrupt. Once the DLE flag is set, the next incoming character is tested for the receipt of the STX character. If it is not received, then the DLE flag is reset and the buffer is clear. If it is the STX character, then the STX flag is set. After this all incoming characters are placed in the I/O buffer, the character count is incremented, and the buffer is tested for any overflow condition. If the buffer overflows, then the IACT task is awakened.

### 3.10.3 Identification of CPC No. 3

This section describes functions performed to process the wind data and produce the desired output data. The routine which performs this function is called "Process" and is identified as CPC No. 3. The Process Task include the routines list in blocks 454 through 527. Process reads in the five-second average received from the sensor and these values are added to the database and used to calculate the wind data output parameter. This includes mean wind direction, mean wind speed, gust, direction variability, gust spread, peak wind direction, speed, and time of occurrence for the most recent 10-minute, 60-minute, and 24-hour periods and the standard deviation of wind direction over the past 10-minute period.

#### 3.10.3.1 CPC No. 3 Major Loop

Process receives input data from the Inter-Assembly Communication Task via the Process Input Buffers, one for each sensor. Process is continuously monitoring the individual input buffer flags which indicate that new data are available. These flags are set by the Inter-Assembly Communication Task. The new X and Y values are checked to verify that they are within a valid range of  $\pm 250$  knots. If the values are correct they are processed. If the values are invalid this is reported to the BIT routine by the Bad Data routine. Invalid data causes the FPI to flash the wind data displays. A counter called Data Counter and referenced by the variable PASSCNT is used to track the number of consecutive valid input data. If the input is invalid for less than sixty seconds the 10-minute database is cleared and the database is re-established over a 10-minute period of consecutive good data reception. If the number of invalid input exceeds 60 seconds (12 samples) the entire 24 hour data base is declared invalid and will be reflected in the BIT status as well as the displayed data, the AWDS data and the printer output data. The sensor status word is also received from the input buffer to be checked by the BIT routines. If the sensor status is not acceptable the input data will be considered invalid. Figure 18 depicts the major loop of Process.

#### 3.10.3.2 CPC No. 3 Wind Data Processing

The various processing tasks are shown in Figure 19. The routine starts out by testing whether the data is from sensor 1, in which case the Time Queue is updated with the latest time to the nearest minute. The progression to the processing will be presented in the following paragraphs. The routine concludes by testing the value of the Data Counter PASSCNT. This counter is used to establish the number of consecutive valid inputs and is used as follows. If more than 10 minutes of valid data has been received, then the database is valid and the recovery flag is reset. If only one minute of valid data has been received, then only the gust spread value is valid. After two minutes of valid data reception, the wind direction and speed is valid. Direction variability, gust, 10-minute peak wind values, and standard deviation of wind direction all require a full 10-minute database. The new calculations for all the wind parameters are placed in the Result Buffer for the given sensor to be accessed by the various output routines.

### 3.10.3.2.1 Direction Reference

Figure 20 shows the relation of wind velocity and the orthogonal components of wind velocity. North is used as the reference axis of rotation and positive rotation is in the clockwise direction. The arrow on the vector in the figure indicates that direction is referenced to the source of the wind (i.e. the direction from which the wind is blowing). Using the defining equations for X and Y given in figure 20 maps the normal Cartesian coordinate system with positive counterclockwise rotation and 0 degree point on the abscissa or X axis to the meteorological coordinate system with positive clockwise rotation and the 0 degree point on the positive vertical axis.

#### 3.10.3.2.1.1 Five Second Vectorial Average

The values transmitted by the sensor are the five-second orthogonal values produced by averaging five one-second orthogonal values of wind data. These values are converted to polar values of direction and speed. If the speed is less than 1 kt the direction is set to 0°. If the speed is 1 kt or greater and the direction is 0° the direction is stored as 360°. Once these adjustments are made the speed and direction is saved in their respective data base queues.

#### 3.10.3.2.1.2 Two and Ten Minute Vectorial Average

The 5 second average speed and direction values are then converted back to orthogonal components and used to maintain a two minute sum of components and a ten minute sum. These running summations are then used to compute the two minute and ten minute average of direction and speed. The equations governing this process are provide in figure 21. The equations for the optional 10-minute time period on recorders only are also given. Mean wind speeds greater than 1 knot of 0° direction will be displayed as 360°. Winds less than 1 kt will be displayed as 0° and 0 speed. The 5-second average conversions along with the two and ten minute average computations are performed by the routines listed in blocks 454 through 465 of the program listing.

#### 3.10.3.2.2 Gust Spread

Figure 22 shows the processing steps for determining the value of gust spread. This is calculated from the latest one-minute period of five-second samples and is processed every five seconds. This routine initializes a MAX variable and a MIN variable and then searches the latest 12 speed inputs for the minimum and maximum value. The gust spread is then the difference between these two values. The routines used in calculating the gust spread are listed on blocks 471 through 473.

#### 3.10.3.2.3 Gusts

Gust are calculated using the latest ten-minute collection of five-second samples of wind speed. There are two routines, one based on two-minute average speed, the other based on a ten-minute average speed. The ten-minute routine can only be executed on the recorder assembly and than only if the 2/10 jumper is set to select the ten minute routines. The Gust routines are listed on blocks 485 through 488.

#### 3.10.3.2.3.1 Gust Using Two-Minute Average Speed

Figure 23 shows the flow chart of the algorithm to determine the maximum gust over the past 10-minutes. Every five-seconds the current value of gust is found testing three condition: 1) that the 2-minute average wind speed is greater than zero, 2) that the 10-minute peak wind exceeds the 2-minute average wind speed by 5 knots, and 3) the gust spread is greater than 10 knots. If any of these conditions is not met, the value of the five-second gust is set to zero. If all the conditions are met, the value of gust is set to the current five-second value of the 10-minute peak wind speed. The five-second value of gust is placed in a queue which contains the past 10 minutes of gust value. The gust queue is then searched to see if there have been any gust conditions over the latest ten minutes (i.e. is there a non-zero entry). If there has been a gust condition the the latest value for the 10-minute peak wind is displayed as the gust. Otherwise, the value for gust is zero.

#### 3.10.3.2.3.2 Gust Using Ten-Minute Average Wind Speed

The algorithm used to determine the value of gust using ten minute wind speed averages is the same as the algorithm based on two minute average with the exception of one condition. The requirement that the gust spread be greater than 10 knots has been dropped. This routine is shown in Figure 24.

#### 3.10.3.2.4 Peak Wind Calculations

Figure 25 provide the chart of the ten-minute peak wind routine. This routine is listed on blocks 474 through 482. The ten minute value of peak wind (PV) is determined every 5 seconds by finding the maximum value in the speed data queue and the corresponding direction and time of occurrence. This new 10 minute value is compared to the value of the 60 minute peak wind (PVH). (There is a 10 minute wait period starting at 55 past the hour implemented by a software counter CNT. This is used to keep the the previous 60 minute peak from carrying over to the next hour.) If the 10 minute peak is equal to or greater than the 60 minute peak value, the new 10 minute peak becomes the 60 minute peak value along with its associated direction and time of occurrence. At 55 minutes past the hour the latest 60 minute peak wind value is output and is also placed in a queue containing the past twenty-four 60 minute peak wind data. The maximum value from this queue is the 24 hour peak wind data (PVD).

#### 3.10.3.2.5 Directional Variability

Direction Variability (DV) is computed every 5 seconds from the 120 five second values stored in the direction data queue. The algorithm used to determine DV is given in Figure 26. This routine tracks the clockwise and counter clockwise rotational changes from one five second interval to the next from an arbitrary starting point. The starting point used is the latest five second input. The rotational comparison are made to consecutive five second values proceeding to the oldest value. A correction is made to any resulting rotation which exceeds  $180^\circ$  between consecutive five second values to allow for crossing the zero pole. The clockwise and counter clockwise rotational extremes are used to determine the final results. This procedure can track through  $360^\circ$ . If a DV of  $360^\circ$  or greater is detected, the current value of the average direction will be displayed in both windows. Examples are provided in Figure 27A and 27B. The listing for the Direction Variability routine is on block 490 through 494.

### 3.10.3.2.6 Standard Deviation of Direction

The standard deviation of wind direction is computed and reported each minute and is based on the previous ten minutes of recorded 5 second sample of wind direction. The algorithm used computes the angle difference between a 5 second sample and the ten minute mean direction. This difference is unsigned and is corrected to be less than 180 degrees to compensate for the zero pole crossing. This produces the deviation of a 5 second sample of wind direction from the ten minute mean wind direction. The 5 second deviation samples are used to produce two averages (1) the average of the deviations squared, (2) the square of the average of the deviations. The difference of these means is the variance of direction. The standard deviation is the square root of the variance. The chart for this routine is provided in figure 28. Blocks 498 through 500 of the listing provide the program code.

### 3.10.4 Identification of CPC No. 4

The following paragraphs describe the Front Panel Interface CPC. This CPC displays wind, time, and system status information on the front panel LED. It also processes control information entered by front panel switches. Charts for this CPC are provide in figures 32 through 38. The program code is listed in blocks 430 through 403.

#### 3.10.4.1 CPC No. 4 Description

There are two functions performed by the FPI program. One function is to output information to the various displays and is performed by the Front Panel Display Task (FPDT). The other function is to receive commands entered via the front panel switches. This is controlled by the Front Panel Switch Handler (FPSH). Since the command entries affect the FPDT, the FPSH will be described first, following a description of some control flags used by these routines.

##### 3.10.4.1.1 CPC No. 4 Front Panel Display Flags

There are a number of different control and status flags which are used to support the front panel display task. Some of these flags are simply a single bit assigned at a memory location (e.g. REFRESH flag and UPDATE flag). Others have more elaborate bit assignments. The SENSOR#, DATE/TIME, ACKNOWLEDGE, and STATUS flags are presented in the following paragraphs.

###### 3.10.4.1.1.1 CPC No. 4 SENSOR# Flag

The bit assignment of this control flag is given in Figure 29. This flag consists of an 8-bit memory location with two groups of 4 bits, assigned separate functions. One group is used to indicate which sensor is the "active" sensor. The other group has an assembly dependent function. It is used to specify which sensor was selected for a "Momentary Sensor Selection" command on an indicator assembly. It also specifies which display mode was selected on a Recorder Assembly. These bits are set by the Front Panel Switch Interrupt Handler (FPSIH) and are read by the Front Panel Display Task (FPDT).

#### 3.10.4.1.1.2 CPC No. 4 CLOCKFLAG Flag

The bit assignment of the CLOCKFLAG flag is shown in Figure 30. Six bits are assigned to functions for setting the clock/calendar. The Field Select bits are used to indicate which time/date parameter has been selected for a setting operation. The UP and DOWN flags indicate whether selected parameters should be incremented or decremented, respectively. There is an additional flag which is not unique to the clock setting function. The FLASH flag is assigned to bit 8 and is used to support display flashing functions. The FLASH flag is toggled every 500 ms by the FPDT. The remaining DATE/TIME flags are set by the FPSIH and are monitored by FPDT.

#### 3.10.4.1.1.3 CPC No. 4 ACKNOWLEDGE/STATUS Flag

This is a group of paired flags used to support the status display. The status display readout reflects the operating conditions of the assembly and the sensors. The STATUS flag bits are used to report certain conditions. (A detailed description of the status display and the bits is given in Paragraph 3.10.7.) The ACKNOWLEDGE flags are used to indicate to the assembly program that the condition has been reported and acknowledged by operations personnel. When a condition is detected, both the status bit and the acknowledge bit are set. The status readout will display a condition code and will flash as long as both bits are set. When the status clear switch is activated, the acknowledge flag is cleared while the status bit is unchanged. This will stop the flashing of the readout, but the condition code will continue to be displayed. These flag states are depicted in Figure 31. The ACKNOWLEDGE flags are reset by the FPSIH. The STATUS flag is set and cleared by the BITS.

#### 3.10.4.1.2 CPC No. 4 Front Panel Switch Handler

Any switch closure will be latched by the switch interface and will generate an interrupt. The interrupt service routine will read the switch latch and perform the function designated to the particular switch position. Since the function of the switch is defined in software, a particular switch can implement one function on an Indicator Assembly and a different function on a Recorder Assembly.

##### 3.10.4.1.2.1 CPC No. 4 Status Clear Switch Function

The routine in Figure 32 starts by checking to see if the STATUS CLEAR button is activated. This switch has been assigned two functions: 1) to inhibit flashing of the STATUS display and 2) to initiate or terminate the STATUS DISPLAY MODE (SDM). If STATUS CLEAR is the only switch detected, then flashing of the front panel STATUS readout is terminated until a new error is detected. If the LAMP TEST switch is also activated along with the STATUS CLEAR switch, then the SDM flag is toggled. If, after toggling, the SDM flag is set, then the SDM is initiated by setting the SENSOR SELECT flag to display the status report starting with sensor 1. If the SDM flag is clear, then the action taken is assembly dependent. An indicator assembly will revert to displaying the active sensor. A recorder assembly will return to normal display mode. Whether initiating or terminating the SDM function, the display information is changed. This change is indicated to the display routine by setting the UPDATE DISPLAY flag.

#### 3.10.4.1.2.2 CPC No. 4 Sensor/Mode Select Switches

The SENSOR SELECT switches of (Figure 33) the indicator perform active sensor selections and momentary sensor display functions. The same switch positions are used for DISPLAY MODE selection on recorder assemblies. When one of these switches is activated, the routine determines if it is a recorder assembly and if so will set the display mode according to which of the four switches was pressed. If the assembly is an indicator, then the switch function is dependent on whether the ESD mode is active. The ESD mode causes these switches to display the status of the corresponding sensor. If the ESD flag is clear, then a test of the ACTIVE SENSOR SELECT switch is conducted and if set the switch number is stored as the active sensor. If the ACTIVE SENSOR SELECT switch is clear, then MOMENTARY SENSOR SELECT function is indicated. A test is made to determine if the same sensor was selected as the MOMENTARY SENSOR, in which case the MOMENTARY SENSOR SELECT function is terminated. If a new switch was activated as a MOMENTARY SENSOR SELECT function, the switch number is saved and a one-minute timeout period is initiated. Since all of these functions require that the displays be updated, the UPDATE DISPLAY flag is set.

#### 3.10.4.1.2.3 CPC No. 4 Clock Control Switches

A set of four (4) switches are provided for setting and enabling the Front Panel clock display (Figure 36). Pressing one of these switches will cause a separate flag to be set. These flags are monitored by the display task which performs the selected function. Consecutive activation of the SET/RUN switch will toggle between these two modes by alternately setting and clearing the SET/RUN flag. The other clock control flags are cleared each time the SET/RUN switch is pressed. Actuating any of the other three (3) switches--FIELD SELECT, UP, or DOWN--causes the respective flag to be set. These three flags are cleared by the Display Task when the function is executed. To describe the switch functions further, assume the clock is in the run mode (i.e. SET/RUN flag is cleared). Pressing the SET/RUN switch will set the corresponding flag and clear the other three switch flags. With the FIELD SELECT pointer cleared, the hours display is selected for a setting function. This will cause the selected field to flash. Each time the FIELD SELECT switch is pressed, the pointer is increased as a modulo 6 counter. The selection sequence is hours, minutes, seconds, month, day, and year. Once a field is selected, the UP and DOWN switch can be used to increment or decrement the displayed value, respectively.

#### 3.10.4.1.3 CPC No. 4 Front Panel Display Task

Figures 35 and 38 provide a detailed diagram of the FPDT program. This program controls all the functions for the display of wind data, status information, time, and date information. The time and date information is updated each second. The wind data and status information is normally updated every five-seconds. Update on demand occurs in response to commands which request the display of new information. Some display modes require various readouts to flash at a 1-Hz rate. To support this flash rate, the displays are refreshed every 500 ms, which means that data is written to the display at a rate of 2 Hz. The 1-Hz flash rate can then be implemented by writing data to the display for 500 ms and then writing a blanking code to the display for the next 500 ms. If no flashing is required, data is written to the displays during both 500-ms periods.

The above has explained the basic function and the timing involved with the FDPT. The following paragraphs will present wind data, status, and time/date display functions.

#### 3.10.4.1.3.1 CPC No. 4 Wind Data Display Functions-Indicator

The task starts out (Figure 35) by testing the REFRESH flag which is set by the scheduler every 500 ms. If this flag is set, the UPDATE flag is tested to determine if new five-(5) second data values should be displayed. Next, the STATUS DISPLAY flag is tested. The function performed, if the flag is set, is described in the next paragraph. Assuming that both the STATUS DISPLAY and UPDATE flags are clear, the next test is to determine if the assembly is a recorder, in which case a request for the latest peak wind data may need to be serviced. If so, pointers to the peak wind data are setup. Otherwise, pointers to the regularly displayed wind data are established. Each value is fetched and tested for validity. (Invalid data has been stored as a negative value.) If the data is invalid and the FLASH flag is set, then a code is sent to blank the affected readout. If the FLASH flag is reset, the value is displayed. If the value is valid, then it is sent to the readout. This process is repeated for each output value. If the UPDATE flag is set, the routine clears the flag and tests the STATUS DISPLAY flag. (Notice that this flag is tested regardless of the path taken as a result of the UPDATE flag.) Assuming that this flag is clear, the update function tests the type of assembly. An indicator assembly will read the sensor number from SENSOR#. If it is a momentary sensor number the corresponding sensor LED will be turned on. Otherwise the LED will be turned off. A recorder assembly will read the display mode from SENSOR#. The appropriate data will then be copied to the display buffer.

#### 3.10.4.1.3.1.1 CPC No. 4 Wind Data Display Functions-Recorder

The recorder assembly display functions differently from that of the indicator (Figure 37B). The readouts are updated once each minute instead of once every five seconds. There are also four display modes that a recorder can operate: 1) NORMAL, 2) PEAK10, 3) PEAK60, and 4) PEAK24. The information displayed in normal mode is the same as for the indicator. The other modes display peak wind data in which the direction and speed of the peak wind are displayed in the WIND DIRECTION and WIND SPEED readouts, respectively. The clock/calendar displays the time of occurrence, while the standard deviation is displayed in the second DIRECTION VARIABILITY window. These modes display the latest 10-minute, 60-minute, and 24-hour peak wind values. Each mode is selected using front panel switches which function as sensor select switches on an indicator assembly.

#### 3.10.4.1.3.2 CPC No. 4 Status Display Function

The status display function (Figures 36A and 36A) is entered by the display task if the STATUS DISPLAY flag has been set by the Front Panel Switch Interrupt Handler. The UPDATE flag may have been set by either the scheduler or the Front Panel switch Interrupt Handler. In either case the SENSOR DISPLAY # is read, the corresponding LED is turned on and the ASSEMBLY STATUS WORD and the selected SENSOR STATUS WORD are copied to the Display Buffer. The assembly status is displayed in the wind direction and wind speed window. The sensor status is displayed in the direction variability window. The status display mode is

distinguished from other display modes by blanking the gust and gust spread displays and by flashing all zeros on the clock display. An explanation of the displayed status words is given in Paragraph 3.10.7 describing the BITS.

#### 3.10.4.1.3.3 CPC No. 4 Clock Setting/Display Function

The clock routine (Figure 38) displays the current time and performs the set functions selected via the front panel switches. Six time/date values are displayed: hours, minutes, seconds, day, month, and year. These values are generated by the real-time clock (RTC). Each value is read from the clock individually. A test is performed on the SET/RUN flag. If it is clear, the time value is saved as part of the time stamp and is displayed. If the SET/RUN flag is set, then a comparison of the FIELD SELECT # and the present time parameter is performed. If there is a match, the UP and DOWN flags are tested and if set the time value is incremented or decremented, respectively. The new value is saved and the FLASH flag is tested. If the flag is set, a blanking code is set to the readout. Otherwise, the new value is displayed. The procedure is followed for each of the six time/date values.

#### 3.10.5 Identification of CPC No. 5

The Automatic Weather Distribution System (AWDS) interface routine is described in this section. The program code for this CPC is listed on blocks 425 through 431.

##### 3.10.5.1 Description of CPC No. 5

The AWDS routine controls the interface and output information as prescribed in Paragraph 3.10.5 of MMA-85B. Figures 39 and 40 show the operations performed to support the AWDS. Output is commenced every 5 seconds with the setting of the AWDS flag by the Scheduler. The CS and DM interface control flags are tested for the set condition. The latest calculated wind parameters are moved to the AWDS buffer for each sensor in the system. These parameters are then transmitted to the AWDS using the message format as specified by Table II of MMA-85B. A separate message is transmitted for each sensor in the system. Figure 52 shows an example message with each field in the message identified.

#### 3.10.6 Identification of CPC No. 6

This section describes the routines which support the printer interface of the recorder assemblies. This program listing of this CPC is covered by blocks 413 through 421.

##### 3.10.6.1 Description of CPC No. 6

The printer routine is shown in Figures 41 and 42. The printer routine is enabled every minute with the setting of PRINTFLAG by the Scheduler. The status register is checked to verify the printer is on-line. If it is not on-line an error will be reported. The wind parameters from the active sensors are moved to the printer buffer. A header is printed on every tenth line. The output format is to print the data and time, then the wind parameters, active sensor number, and system status. As each wind parameter is printed it is tested for validity. If a parameter is invalid, a star is printed to the right of the invalid value.

### 3.10.7 Identification of CPC No. 7

The following paragraph describes the Built In Test (BIT) routines. Three BITs are separate and independent programs which test the CPU, ROM, and RAM. There are several other BITs which are part of other CPCs. These will be described in this section and cross referenced to the corresponding CPC. The method of reporting the status of these tests is also presented. The charts for the BIT routines are provided in figures 45 through 49. Program code listing for the BIT task is provided in blocks 405 through 411.

#### 3.10.7.1 CPC No. 7 Description

The BITs are a set of self test routines which are designed to monitor assembly performance and report and detect faults which may degrade performance. These tests are run continuously as a part of normal assembly operations. Each test is repeatedly performed several times each second. If a fault condition is detected the routine sets a bit in a RAM location and the Indicator/Recorder Status word. Figure 43 shows the organization of this status word. The information from this status word and a similar one for the sensor are combined to form a byte called the General Status Byte. This byte determines the value normally displayed by the FPI routine. Figure 44 shows the organization of the status byte. The following paragraph describes the test routines.

##### 3.10.7.1.1 CPC No. 7 BIT Error Code Display Functions

The results of the BIT are displayed using a two tier reporting scheme. The first level is presented by the two LEDs of the status display. This display gives a general report of status of the assembly and the system. The error codes for this display are given in Tables XII and XIII. The second level of error reporting is accessed by holding the LAMP TEST switch down and depressing the STATUS CLEAR switch. This will place the assembly in the Status Display Mode which will report the results of all the BITs using several of the display LEDs as indicated in the Tables. These display codes in the Status Display Mode are referenced by code group numbers. This includes the results of the BITs performed on the sensors. The error codes for the Status Display mode are given in Tables XIV through XXI. Figure 53(A) provides a view of the front panel identifying the status codes for the BIT performed on the Indicator/Recorder and the BIT codes for the the sensor. Figure 53(B) identifies the location of each code group.

##### 3.10.7.1.1.1 CPC No. 7 Status Display Acknowledge Function

The general status display will normally read out a constant 00 indicating no errors. When an error is detected, an error code will start flashing in the display. The error condition can be acknowledged by pressing the status clear switch. This will inhibit flashing of the display, but will maintain the value of the error code if the error is still present.

TABLE XII  
 GENERAL STATUS CODES  
 STATUS DISPLAY-LEFT DIGIT

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Error
1	Processor Board Fault Failure of one or more of the following tests  1) CPU Test 2) ROM Test 3) RAM Test 4) Inter-Assembly Communication Port Loop Test
2	Sensor Error-Any error report from the sensor
3	Combination of codes 1 and 2
4	Loss of Master
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

**TABLE XIII**  
**GENERAL STATUS CODES**  
**STATUS DISPLAY-RIGHT DIGIT**

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	Printer Error (Recorder Only)
2	Inter-Assembly Communication Error 1) Communication Timeout and/or 2) Multiple CPC Error 3) Carrier Timeout Error 4) No Sensor Response Error
3	Combination of codes 1 and 2
4	Recovery in Progress
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

TABLE XIV

STATUS DISPLAY MODE CODES FOR  
THE INDICATOR/RECORDER STATUS WORD  
WIND DIRECTION DISPLAY-LEFT DIGIT (CODE GROUP 1)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	Inter-Assembly Communication Loop Test Failure
2	Inter-Assembly Communication Time Out Error
3	Combination of code 1 and 2
4	Inter-Assembly Communication Carrier Timeout Error
5	Combination of Code 1 and 4
6	Combination of code 2 and 4
7	Combination of code 1, 2, and 4

TABLE XV

STATUS DISPLAY MODE CODE FOR  
THE INDICATOR/RECORDER STATUS WORD  
WIND DIRECTION DISPLAY-RIGHT DIGIT (CODE GROUP 2)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	CPU Test Failed
2	ROM Test Failed
3	Combination of code 1 and 2
4	RAM Test Failed
5	Combination of Code 1 and 3
6	Combination of Code 2 and 4
7	Combination of Code 1, 2, and 4

TABLE XVI

STATUS DISPLAY MODE CODES FOR  
THE INDICATOR/RECORDER STATUS WORD  
WIND SPEED DISPLAY-LEFT DIGIT (CODE GROUP 3)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Error
1	Printer Paper Out
2	Printer Off-Line
3	Combination of codes 1 and 2
4	Printer Fault
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

TABLE XVII

STATUS DISPLAY MODE CODES FOR  
THE INDICATOR/RECORDER STATUS WORD  
WIND SPEED DISPLAY-RIGHT DIGIT (CODE GROUP 4)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	AWDS Loop Test Failed
2	N/A (Not Assigned)
3	N/A
4	Loss of Master
5	Combination of codes 1 and 4
6	N/A
7	N/A

TABLE XVIII

STATUS DISPLAY MODE CODES FOR  
THE SENSOR STATUS WORD  
DIRECTION VARIABILITY LEFT DISPLAY-LEFT DIGIT (CODE GROUP 4)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	Multiple Reset Error
2	Counter Test Error
3	Combination of code 1 and 2
4	Inter-Assembly Loop Test Error
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and

TABLE XIX

STATUS DISPLAY MODE CODES FOR  
THE SENSOR STATUS WORD  
DIRECTION VARIABILITY LEFT DISPLAY-RIGHT DIGIT (CODE GROUP 6)

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	CPU Test Failed
2	ROM Test Failed
3	Combination of codes 1 and 2
4	RAM Test Failed
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

**TABLE XX**  
**STATUS DISPLAY MODE CODES FOR**  
**THE SENSOR STATUS WORD**  
**DIRECTION VARIABILITY RIGHT DISPLAY-LEFT DIGIT (CODE GROUP 7)**

<u>Display Code Number</u>	<u>Indicated Status</u>
0	No Errors
1	A/D Test Failed
2	Element Driver Test Failed
3	Combination of codes 1 and 2
4	Temperature Sensor Test Failed
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

**TABLE XXI**  
**STATUS DISPLAY MODE CODES FOR**  
**THE SENSOR STATUS WORD**  
**DIRECTION VARIABILITY RIGHT DISPLAY-RIGHT DIGIT (CODE GROUP 8)**

0	No Errors
1	Out of Calibrated Range Error
2	Pressure Sensor Test Error
3	Combination of codes 1 and 2
4	Power Supply Test Error
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

TABLE XXII

STATUS DISPLAY MODE CODES FOR  
THE SENSOR STATUS WORD  
GUST SPREAD DISPLAY-RIGHT DIGIT (CODE GROUP 9)

0	No Errors
1	Long Recovery in Progress
2	No Sensor Response
3	Combination of codes 1 and 2
4	CRC-16 Error
5	Combination of codes 1 and 4
6	Combination of codes 2 and 4
7	Combination of codes 1, 2, and 4

#### 3.10.7.1.2 CPC No. 7 Description of the CPU Test

The CPU test performs a sequence of operations on the four accumulators of the 8088 microprocessor, AX, BX, CX, and DX. The results of these operations are checked for validity. Detection of any error will be recorded by setting bit 0 of the Indicator/Recorder Status Word. Figure 45 shows a flow graph of the CPU test. Hexadecimal constants are loaded into the accumulators. AX and BX are summed to form FF which is left in AX. The 55 in CX is subtracted from the FF in AX with the resulting AA being stored in CX. CX and DX are summed to form FF which is stored in DX. This result is compared with the correct result, thereby testing the accumulator and the zero flag of the CPU status register. A test is also performed on the sign flag and carry flag.

#### 3.10.7.1.3 CPC No. 7 Description of the ROM Test

The ROM test is accomplished by performing a 16-bit summation of all the locations of the EPROM to form a checksum. This value is compared with the correct checksum value which was produced and saved in the EPROM when it was programmed. The difference between these sums will cause the setting of bit 1 of the IRSW. Figure 46 shows the flow chart of this routine.

#### 3.10.7.1.4 CPC No. 7 Description of the RAM Test

The RAM test is accomplished by writing a test pattern to a RAM location and verifying that the same pattern is read back. This test is performed on all RAM locations. The data in each location is saved before testing and restored at the completion of the test. Interrupts are disabled to maintain complete control of RAM access during this test. For this reason only small sections of RAM are tested during each iteration. Figure 47 shows a graph of this routine. Two test patterns are used at each location, a 55 pattern and an AA pattern. Both are comprised of alternating ones and zeros and are used to detect bit locations which are no longer functioning.

#### 3.10.7.1.5 CPC No. 7 Description of Inter-Assembly Communication Loop Test

The Inter-Assembly Communication Loop Test is performed each time an assembly sends an inter-assembly message. When the first character of a message is transmitted, the receive circuitry is enabled in order to read the character back. The received character is compared to the transmitted character. Bit 4 of the IRSW is set if the characters are different. Figures 48A and 48B is a flow chart of this test operation.

#### 3.10.7.1.6 CPC No. 7 Description of Inter-Assembly Timeout Error

The master assembly normally polls the sensors every five seconds. If there are no polls or sensor responses over a period greater than five seconds then this will cause a timeout error. This function is provided by the Inter Assembly Task described in Paragraph 3.10.2 and shown in Figure 11. Bit 5 of the IRSW is set to indicate a timeout error.

### 3.10.7.1.7 CPC No. 7 Description of Inter-Assembly Multiple CRC Error

All inter-assembly communications are checked for validity through the use of a CRC-16 number which is appended to the end of each message. The Inter-Assembly Communication Task verifies the CRC-16 of a message when it is received. Each time a message is received with a bad CRC number, a counter is incremented. If the bad CRC count is equal to or greater than 3, an error is recorded by setting bit 6 of the IRSW.

### 3.10.7.1.8 CPC No. 7 Description of AWDS Loop Test

The AWDS Loop Test is performed each time an AWDS message is sent. The first character of each message is looped back to the receiver. The received character is compared to the transmitted character. If these are different, bit 8 of the IRSW will be set to indicate a test failure. Figure 49A and 49B provides a flow chart of this test.

### 3.10.7.1.9 CPC No. 7 Description of Printer Error Routines

The printer BIT is performed each time an output to the printer is commenced. The test consists of reading the printer status register and reporting any error conditions that are indicated. These errors include printer-offline, paper-out, and printer fault.

### 3.10.7.2 CPC No. 7 Interfaces

The BITs described above are incorporated in several CPCs and in many cases functions as general error detection routines. Hence a BIT is often associated with the CPC function being test. These are listed in Table XXIII below. The three processor oriented tests are the CPU test, ROM test, and the RAM test and are grouped together with the watchdog timer routine.

TABLE XXIII

DISTRIBUTED BUILT IN TESTS

Task Name	Associated BIT
Inter Assembly Task	Inter Assembly Loop Test
Inter Assembly Task	Inter Assembly Timeout Error
Inter Assembly Task	Inter Assembly Multiple CRC Error
AWDS Task	AWDS Loop Test
Printer Task	Printer No Acknowledge
Printer Task	Printer Off Line
Printer Task	Printer Paper Out
Front Panel Task	Status Code Display

### 3.10.8 Identification of CPC No. 8

The Start Up and Recovery (STUR) CPC is described in this section. This routine is the entry program after a power up or reset condition. The chart for this cpc is provided in figure 50. The program code is listed in blocks 605 through 636.

#### 3.10.8.1 Description of Start Up Modes

There are three starting modes for this program; (1) cold start, (2) warm start, (3) hot start. The difference between these modes is the extent of the data base initialization. The program determines the type of start up mode to use based on the status of the variable POWERFLAG and the difference between the time at start up and the latest time stamp saved in memory. This time stamp is routinely saved during normal program execution. If the battery back-up link is installed as it would be for normal operation the contents of memory are unchanged after a loss of power and the real-time clock will continue to keep time. These attributes are used by the STUR routine as follows:

**Cold Start** - used for initial power up or after power outages longer than one minute. The variable POWERFLAG is used to determine if the the memory has been powered up and initialized at some time in the passed. This variable must contain the decimal value 12345 if the memory was been initialized and has been made non-volatile by the battery. If this value is not present the a Cold Start process is commenced to initialize (zero) all of the RAM starting above the vector tables and continuing to the top of the RAM space used by the program. The same initialization is used if the difference between the real-time clock value at power up and the last recorded time stamp is more than 60 seconds.

**Warm Start** - used to initialize only the ten minute data base, saving the portion of the data base containing the 24 hour results. If the difference between the power up time and the time stamp is less the 60 second but greater than 5 second a warm start is executed.

**Hot Start** - used to maintain the data base after momentary interruptions of power of less than five seconds.

These different modes are used to preserve the data base and prevent rebuilding the entire data base after momentary interruptions of power.

#### 3.10.8.2 Description of CPC No. 8

The Start up and Recovery routine first step is to initialize the interrupt vector and the task tables in RAM and proceeding to initialize all the hardware interface devices. The POWERFLAG variable is tested to determine if a cold start is required. If a cold start is not indicated by this test the time stamp is compared to the current time of the RTC. If this comparison indicated a time difference greater than 60 second a cold start is initiated. If the time difference is less than 60 second but greater than 5 seconds a warm start is commenced to initiate the ten minute data base values. If the time difference is five seconds or less then normal processing is resumed after validating the the data base. This is accomplished by checking that the running sums for the speed and direction queues maintained during normal data processing are the same as the sum of the queue contents. If the database is invalid a rebuild operation will commence at the completion of STUR.

After determining the type of start up mode and initializing the data base accordingly execution of the start up routine progresses through the following steps for all cases. Variables used as flags are initialized, with the five second counter variable 5SECCTR being set to 5 , the POWERFLAG being set to 12345, the display task flag DISPFLAG being set to 1 and all remaining flags being cleared. The program then reads the configuration status of each jumper and saves the results in the variable CONFIG. The program then initializes the front panel switch LEDs and set the variable CONTRLFLAG if the master mode is enabled. The data base of unused sensors is initialized to display a flashing "55" if selected. The STUR routine concludes by enabling each task and enabling the interrupts then calling the Display Task.

### 3.11 PROGRAM LISTINGS

Program listings are provided in a separate bound section titled " Program Listings for the AN/FMQ-13(V) Digital Indicator/Recorder Application Program"

#### 3.11.1 Naming Conventions

The following naming conventions are used throughout the program listing. Attributes for name differentiation include upper and lower case lettering and hyphenated names or names containing a period.

FORTH defined words (i.e. those words defined as part of the standard Forth programming language) use capital letters or symbols (e.g. IF, THEN, ELSE, DO, LOOP, =, 0=, !, @). These are defined in the "polyFORTH II Reference Manual". Forth assembly code language is also represented in upper case letters and may include duplicate names for some words (e.g. IF, THEN, ELSE, DO, LOOP). These are defined in the "INTEL 8086 CPU Supplement to the polyFORTH II Reference Manual".

All variable names are upper case with no hyphens or periods. Application program words (the names of routines) are upper case and use a period or hyphen in the name (e.g. GET.TIME, GET.DATE, TIME.STAMP, ?.#SENSORS). Constants used to reference hardware device registers are lower case (e.g. clock, clock/csr, comm/data). Assembly code routine names are lower case with a symbol or a period within the name (e.g. set.status, clear.status, clock>, >clock).

The one exception to these naming conventions is for the data structure programs used to access the queues of the data base. These program names are used as variable names in the various routines that reference the data base and appear as upper case variables names (e.g. SPEEDQ, ANGLEQ, GUSTQ). These words are listed on blocks 347 through 351.

#### 3.11.2 CPC Block Numbers

A list of the CPC names and the corresponding listing blocks are provided in Table XXIV.



#### 4. QUALITY ASSURANCE

This section provides a review of the test plans and procedures used to qualify the CPCI. Qualification was performed on the built-in-test function during the maintainability demonstration. The remaining functions were qualified during performance testing.

##### 4.1 TEST PLAN/PROCEDURE CROSS REFERENCE INDEX

Table XXV provides a list of the CPCI functions cross referenced to the test procedure used for qualification. The table references maintainability testing and performance testing. The maintainability demonstration was conducted in accordance with the Maintainability Demonstration Plan document # 9131-1008 (CDRL Item Q005) the results of which were reported in the Maintainability Demonstration Report document number WMS-MDR-01 (CDRL Item Q006). The performance test were conducted in accordance with the test procedure titled "First Article Performance Test for Wind Measuring Set AN/FMQ-13(V)" document # 9131-1005 (CDRL Item K002). This test procedure was developed in accordance with the "Equipment Test Plan for Wind Measuring Set AN/FMQ-13(V), document # WMS-ETP-01-R1 (CDRL Item K001). Performance testing of the wind data processing required the use of a simulation program to provide known inputs to the indicator/recorder program while recording the output results. The test concept for the use of this simulation is provided below.

##### 4.1.1 Performance Test Simulation Test Concept

An IBM PC computer, or equivalent (herein referred to as the sensor simulator) is used to simulate the sensor assembly while performing the detailed test procedures of paragraph 6.2.4. Wind data processing is verified by monitoring the output values (both the LED displays and AWDS values) to a known input and applying the definitions for each parameter as given in paragraph 3.2.1.3.6 of the specification regarding wind data processing. The expected value of any parameter can be calculated and used to verify the recorded output. The sensor simulator produces ten sets of known wind direction and speed settings. The values are given in the table below. This produces nine step changes as inputs to the indicator with each setting held constant for three minutes. This allows the two-minute average direction and speed values to converge to the setting for two minutes and remain constant for one minute. The last setting is held constant until the simulator is turned off.

A few examples of expected values of various wind parameters follow:

- (1) At the end of the first three minute input setting period all parameters will be zero.
- (2) At the beginning of the second setting (90° and 9 knots) the gust spread should be 9 knots and should remain at nine knots for one minute. Since the gust spread is below 10 knots the gust value should be zero. The ten minute peak wind value should be 90° and 9 knots with a time of occurrence of the present time of the AWDS message. The average speed will be zero on the first reading ( $1/24 \times 9 = 0.375$  knots which is displayed as zero). On the second output of the second setting the average speed will 1 knot ( $2/24 \times 9 = 0.75$  knots which is rounded to 1 knot).

TABLE XXV

FUNCTION VS TEST CROSS INDEX

FUNCTION	TEST
SEQUENCE CONTROL	PERFORMANCE
I/O CONTROL	PERFORMANCE
DATA PROCESSING	PERFORMANCE
DISPLAYS	PERFORMANCE
OPERATIONAL CONTROL	PERFORMANCE
ERROR DETECTION	MAINTAINABILITY
REAL-TIME DIAGNOSTICS	MAINTAINABILITY
FAULT RECOVERY	PERFORMANCE

SENSOR SIMULATOR		
<u>DIRECTION AND SPEED SETTINGS</u>		
<u>SETTING #</u>	<u>DIRECTION</u>	<u>SPEED</u>
1	0	0
2	90	9
3	180	20
4	270	50
5	300	100
6	330	150
7	355	200
8	10	250
9	180	50
10	350	10

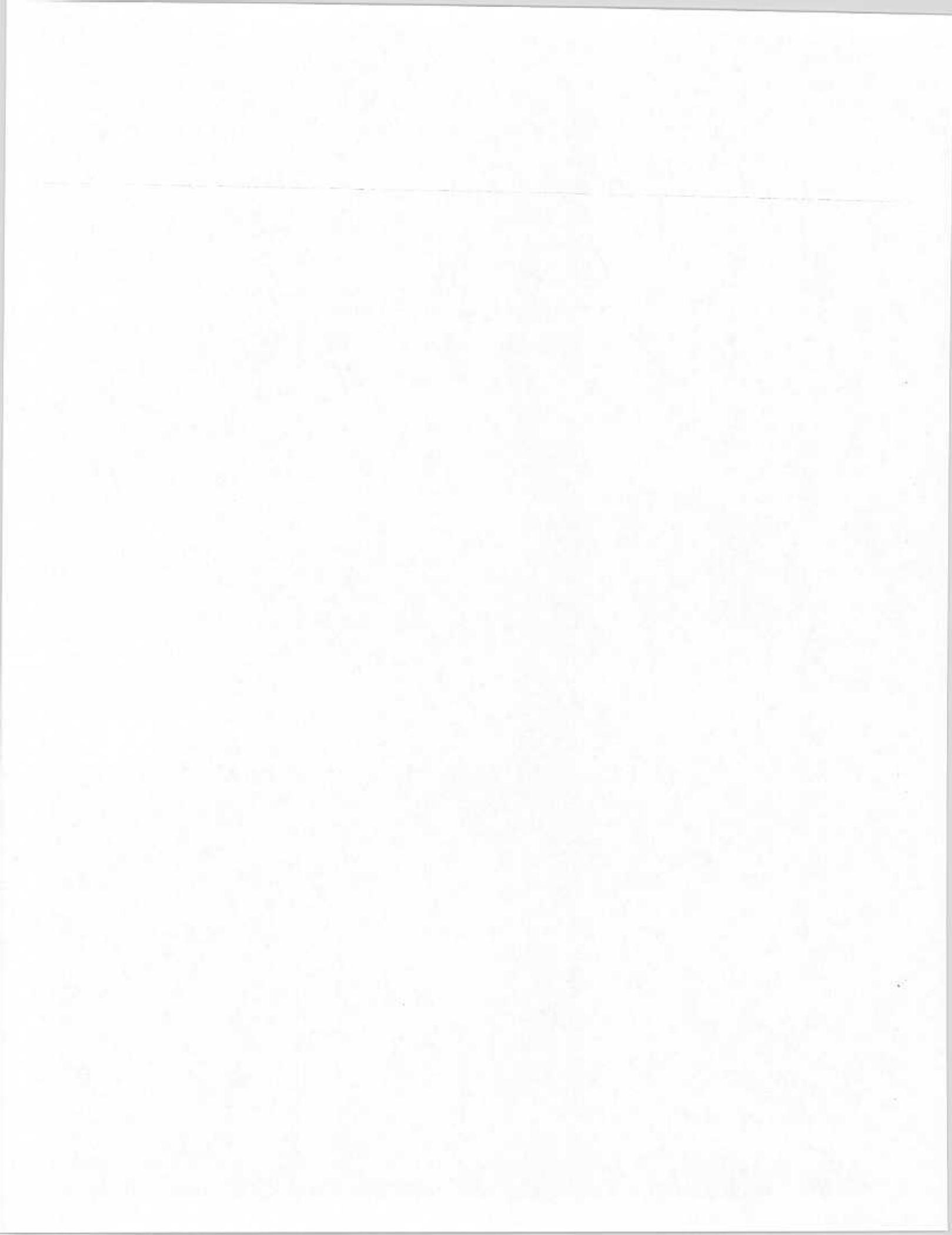
#### 5. PREPARATION FOR DELIVERY

Delivery preparation for this CPCI is specified in the CDRL by items E002, E004, and E00D specifying "Software Delivery Documentation", "Version Description Document", and "Computer Software/Computer Program/Computer Data Base Configuration Item", respectively. The computer program is titled "Digital Indicator/Recorder Application Program". Two copies of the source code and one copy of the object code are provided on 5.25 inch diameter floppy disks which are MS-DOS format compatible. One paper copy of the source code and object code are also specified. The disks and the paper copies will be labeled with the system nomenclature (FMQ-13(V)), the program title, release revision and the CPIN for the program.



---

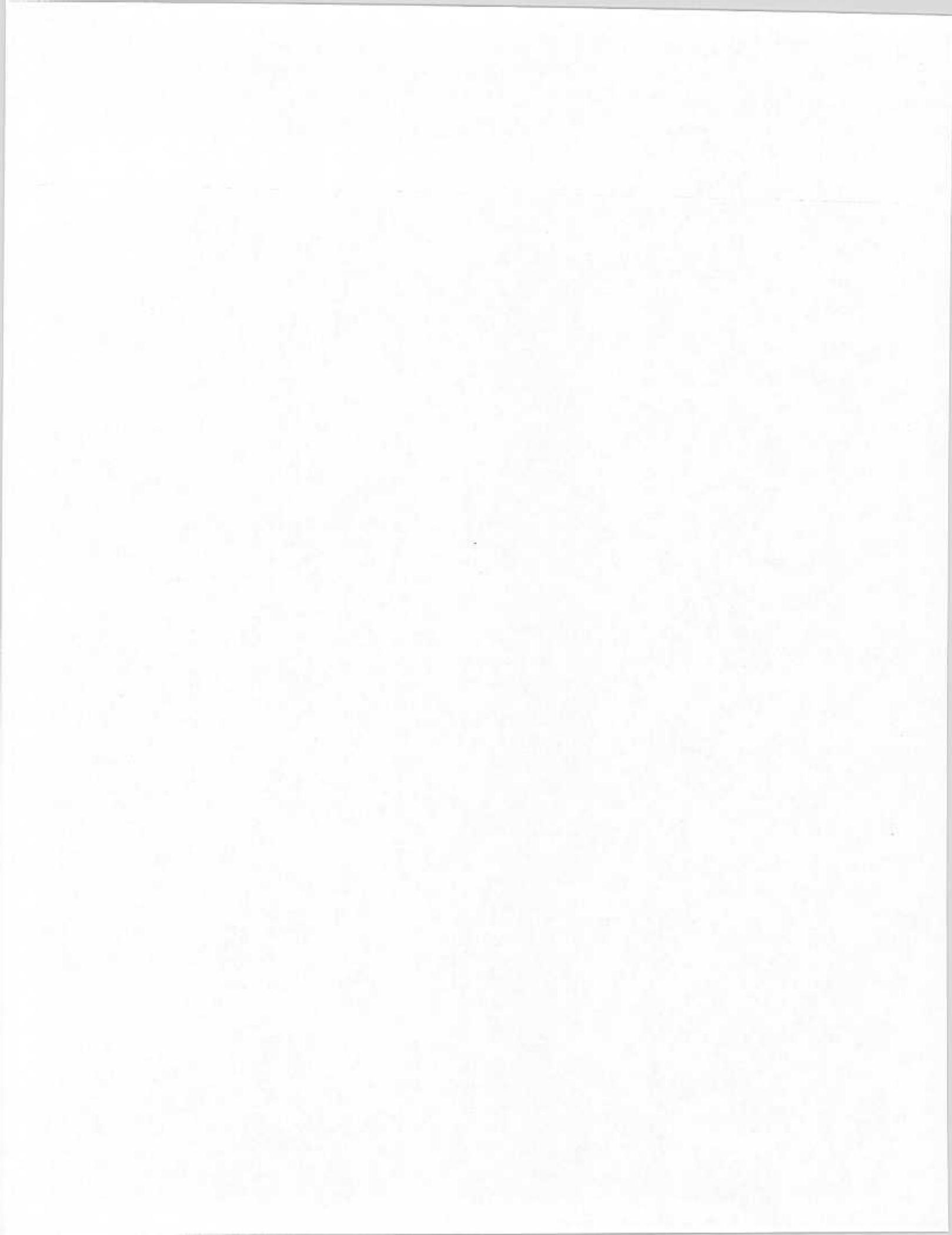
**SECTION SIX  
DATA SHEETS FOR THE  
HARDWARE DEVICES**



---

7170

REAL-TIME  
CLOCK INTERFACE





ICM7170  
µP-Compatible  
Real-Time Clock  
PRELIMINARY

GENERAL DESCRIPTION

The ICM7170 real time clock is a microprocessor bus compatible peripheral, fabricated using Intel's silicon gate CMOS LSII process. An 8-bit bidirectional bus is used for the data I/O circuitry. The clock is set or read by accessing the internal separately addressable and programmable counter from 1/100 seconds to years. The counters are controlled by a pulse train divided down from a crystal oscillator circuit, and the frequency of the crystal is selectable with the on-chip command register. An extremely stable oscillator frequency is achieved through the use of an on-chip regulated power supply.

The device access time (t<sub>acc</sub>) of 300ns eliminates the need for any microprocessor wait states or software overhead. Furthermore, the ALE (Address Latch Enable) input is provided for interfacing to microprocessors with a multi-bus address/data bus. With these two special features, the ICM7170 can be easily interfaced to any available microprocessor.

The ICM7170 generates two types of interrupts. The first type is the periodic interrupt (i.e., 100Hz, 10Hz, etc.) which can be programmed by the internal interrupt control register. It provides 7 different output signals. The second type is the alarm interrupt. The alarm time is set by loading an on-chip 81-bit RAM that activates an interrupt output through a comparator. The alarm interrupt occurs when the real time counter and alarm RAM time are equal. A status register is available to indicate the interrupt source.

An on-chip Power-Down Detector eliminates the need for external components to support the battery back-up function. When a power-down or power failure occurs, internal logic switches the on-chip counters to battery back-up operation. Input/output and read/write functions become disabled and operation is limited to time-keeping and interrupt generation, resulting in low power consumption.

Internal latches prevent clock roll-over during a read cycle. Counter data is latched on the chip by reading the 100th-seconds counter and is held indefinitely until the counter is read again, assuring a stable and reliable time value.

FEATURES

- 8-bit µP Bus Compatible
- Multiplexed or Direct Addressing
- Binary Time Data Format Lowers Software Overhead
- Time From 1/100 Seconds to 99 Years
- Software Selectable 12/24 Hour Format
- Latched Time Data Ensures No Roll-Over During Correction
- Full Calendar With Automatic Leap Year
- On-Chip Battery Backup Switchover Circuit
- Access Time Less Than 300ns
- Programmable Crystal Oscillator Frequencies
- On-Chip Alarm Comparator and RAM
- Interrupts from Alarm and 6 Selectable Periodic Intervals
- Standby Micro-Power Operation: 2µA Typ. at 3.0V and 32kHz Crystal

APPLICATIONS

- Portable and Personal Computers
- Industrial Control Systems
- Data Logging
- Point Of Sale

ORDERING INFORMATION

PART NUMBER	TEMPERATURE RANGE	PACKAGE
ICM7170UD	-40°C to +85°C	24-PIN PLASTIC DIP
ICM7170JG	-40°C to +85°C	24-PIN CERDIP

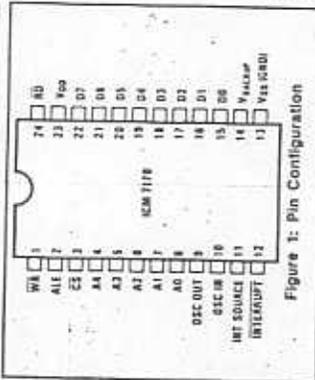


Figure 1: Pin Configuration

**ICM7170**

**ABSOLUTE MAXIMUM RATINGS**

Supply Voltage	0V to +5V
Power Dissipation (Note 1)	500mW
Input Voltage (Any Terminal) (Note 2)	VDD + 0.3V to VSS - 0.3V
Operating Temperature	-40°C to +85°C
Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10sec)	260°C

NOTE 1: Due to the SCR structure inherent in the CMOS process, connecting any terminal at voltages greater than VDD or less than VSS may cause destructive latch-up. For this reason, it is recommended that no output from any ICM7170 be connected to any other ICM7170 pin. VDD and VSS should be connected to the device before the supply is introduced, and that all multiple supply systems, the supply to the ICM7170 be turned on last. Stress above the maximum ratings may cause permanent damage to the device. These are stress ratings only and functional operation of the device is not guaranteed above these limits. Absolute maximum ratings are shown in the operational sections of the specifications if not implied. Exposure to absolute maximum ratings conditions for extended periods may affect device reliability.

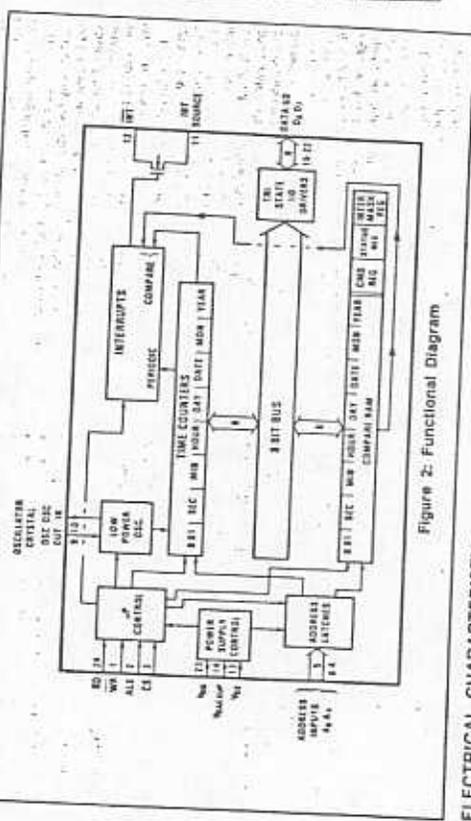


Figure 2: Functional Diagram

**ELECTRICAL CHARACTERISTICS**

D.C. CHARACTERISTICS (TA = -40°C to +85°C, VDD = +5V ± 10%, VBACKUP = VDD - VSS = 0V unless otherwise specified)

SYMBOL	PARAMETER	TEST CONDITIONS	SPECIFICATION			UNIT
			MIN	TYP	MAX	
VDD	VDD Supply Range (Operating)		2.8		5.5	V
IStandby	Standby Current	FATAL = 10MHz PWR 1 & 15-22 & 24 = VDD VDD = VSS, VBACKUP = VDD - 3.0V		2.0	20	µA
IStandby	Standby Current	FATAL = 40MHz PWR 1 & 15-22 & 24 = VDD VDD = VSS, VBACKUP = VDD - 3.0V		20	150	µA
IOS	Operating Supply Current	FATAL = 10MHz PWR 1 & 15-22 & 24 = VDD VDD = VSS, VBACKUP = VDD - 3.0V Read/Write Operation at 100kHz		0.3	1.2	mA
IIOH	Operating Supply Current	FATAL = 20MHz PWR 1 & 15-22 & 24 = VDD VDD = VSS, VBACKUP = VDD - 3.0V Read/Write Operation at 100kHz		1.0	2.0	mA
VIL	Input low voltage	VDD = 4.5V	3.8		0.8	V
VOL	Output low voltage	VDD = 4.5V IOS = 1.0mA			0.4	V

**ICM7170**

**ELECTRICAL CHARACTERISTICS (CONT.)**

SYMBOL	PARAMETER	TEST CONDITIONS	SPECIFICATION			UNIT
			MIN	TYP	MAX	
VDD	Output high voltage except INTERRUPT	IOS = 400µA VDD = VDD or VSS	2.4			V
IOL	Input leakage current	VDD = VDD or VSS	-10	0.3	+10	µA
VBAT1	Backup Battery Voltage	VDD = VDD or VSS	-10	0.3	+10	µA
VBAT2	Backup Battery Voltage	FATAL = 1.2 MHz FATAL = 20MHz	2.8		3.2	V
VOL	Output low voltage	VDD = VDD or VSS IOS = 1.0mA			0.4	V
VOL	Leakage current	INTERRUPT VDD = VDD or VSS			10	µA

AC CHARACTERISTICS (TA = -40°C to +85°C, VDD = +5V ± 10%, VBACKUP = VDD Load Capacitance = 150pF, VA = 0.4V, VIN = 3.5V unless otherwise specified)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
<b>WRITE CYCLE TIMING</b> tWA ADDRESS valid to WRITE strobe tAA ADDRESS HOLD time for WRITE tCS WRITE pulse width, low tD DATA IN to WRITE set up time tD DATA IN hold time after WRITE tCS WRITE cycle time	100 0 100 100 30 400	100 0 100 100 10 400	ns ns ns ns ns ns		
<b>MULTIPLEXED MODE TIMING</b> tA ALE pulse width, high tAH ADDRESS to ALE set up time tA ADDRESS hold time after ALE	30 30	30 30	ns ns ns		

ICM7170

ICM7170

INTELSIL

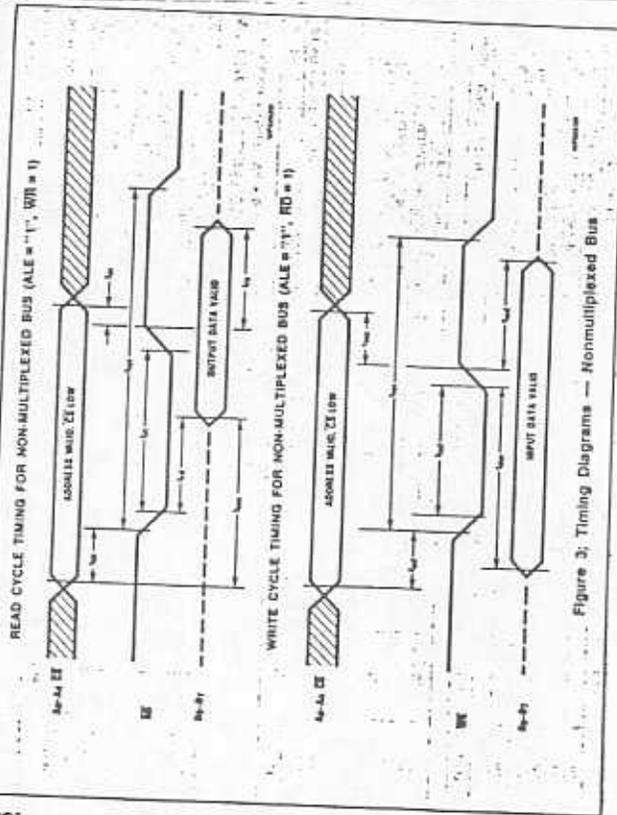
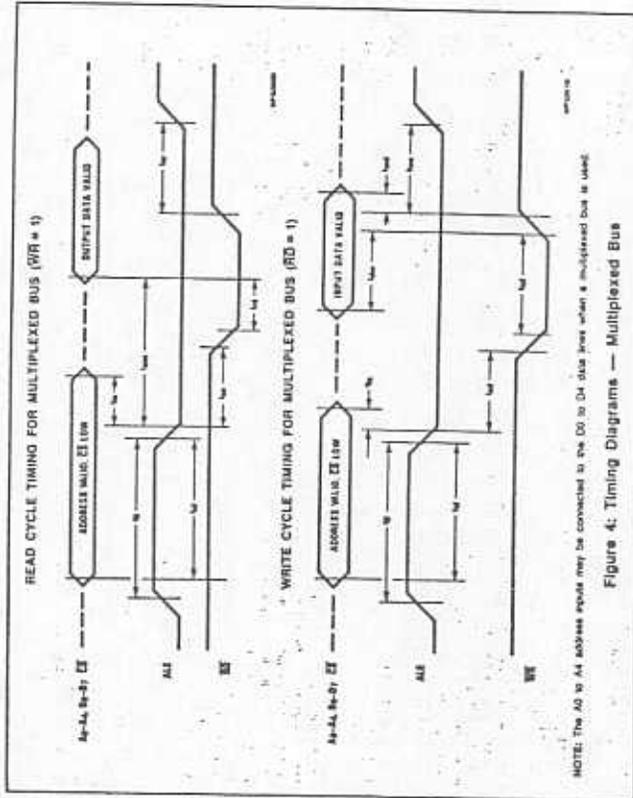


Figure 3: Timing Diagrams — Nonmultiplexed Bus

ICM7170

ICM7170

INTELSIL



NOTE: The A0 to A1 address inputs may be connected to the D0 to D1 data lines when a multiplexed bus is used.

Figure 4: Timing Diagrams — Multiplexed Bus

**Table 1**

SIGNAL	PIN	DESCRIPTION
WR	1	Write input
ALE	2	Address latch enable input
CS	3	Chip select bar input
A1-A0	4-8	Address inputs
OSC OUT	9	Oscillator output
OSC IN	10	Oscillator input
INT SOURCE	11	Interrupt common
INTERMPT	12	Interrupt output
VSS(GND)	13	Digital common
VBACKUP	14	Battery negative side
DD-D7	15-22	Data I/O
VDD	23	Positive digital supply
RD	24	Read input

**DETAILED DESCRIPTION**

**Oscillator**

The circuit uses a standard CMOS Pierce oscillator, for maximum accuracy, stability, and low-power consumption. Externally, one crystal and two capacitors are required. One of the capacitors is variable and is used to trim or tune the oscillator output. Typical values for these capacitors are C1R = 10pF and COUT = 10 - 35pF, or approximately double the recommended CLOAD for the crystal being used. Both capacitors must be connected from the respective oscillator pins to VDD for maximum stability.

The oscillator output is divided down to 4000Hz by one of four selected ratios, via a variable prescaler. The ICM7170 can use any one of four different, low-cost, crystals: 4.194304MHz, 2.097152MHz, 1.048576MHz, or 32.768kHz. The command register must be programmed for the frequency of the crystal chosen, and this in turn will determine the prescaler's divide ratio.

Command Register frequency selection is written to the D0 and D1 bits at address 11H and the 12 or 24 hour format is determined by bit D2, as shown in Table 4.

The 4000Hz signal is divided down further to 100Hz, which is used as the clock for the counters. Time and calendar information is provided by 8 consecutive addressable, programmable counters: 100ths of seconds, seconds, minutes, hours, day of week, date, month, and year. The data is in binary format and is configured into 8 bits per digit. See Table 4 for address information. Any unused bits are held at logic "0" during a read and ignored during a write operation.

**Compare Interrupts**

On the chip are 51 bits of Alarm Compare RAM grouped into words of different lengths. These are used to store the time, ranging from 100ths of seconds to years, for comparison to the real-time counters. Each counter has a corresponding RAM word. In the Alarm Mode an interrupt is generated when the current time is equal to the alarm time. The RAM contents are compared to the counters on a word by word basis. If a comparison to a particular counter is unnecessary, then the appropriate "M" bit in Compare RAM should be set to logic "1".

The "M" bit, referring to Mask bit, causes a particular RAM word to be masked off or ignored during a compare. Table 4 shows the addresses and Mask bit information.

**Periodic Interrupts**

The interrupt output can be programmed for 6 periodic signals: 100Hz, 10Hz, once per second, once per minute, once per hour, or once per day. The 100Hz interrupt, 10Hz interrupt, and the hundreds of a second counter have instantaneous time delays of  $\pm 2.5%$ ,  $\pm 0.15%$ , and  $\pm 2.5%$  respectively; the time delays, however, is zero. These can occur concurrently except, however, in zero interrupts. Both the Periodic and the Alarm interrupts are controlled by the Interrupt Mask Register. The desired interrupt is enabled by writing a logic "1" to the desired appropriate bit, as shown in Table 5.

The Interrupt Status Register, when read, indicates the cause of the interrupt and reads itself on the trailing edge of the read pulse. Once one or more bits have been set in the Mask Register, a logic "1" in a corresponding counter will strobe the appropriate bit in the Interrupt Status Register. The interrupt pin (#12) is then pulled to the potential of the interrupt source pin (#11) through an internal open-drain N-channel MOSFET. This facilitates wire-ORing the ICM7170 with other interrupt generators that must be connected to the system MPU.

**Table 2: Command Register Format**

COMMAND REGISTER ADDRESS (100010, 11H) WRITE-ONLY		D7	D6	D5	D4	D3	D2	D1	D0
Bit	Pin	12	11	10	9	8	7, 2, 4	3	1

**Table 3: Command Register Bit Assignments**

D7	D6	D5	D4	D3	D2	D1	D0	TEST BIT
0	0	27.768kHz	0	17 hour mode	0	Interrupt disabled	0	Normal Mode
0	1	1.048576MHz	1	24 hour mode	1	Interrupt enable	1	Test Mode
1	0	2.097152MHz	0	12 hour mode	0	Interrupt disabled	0	Normal Mode
1	1	4.194304MHz	1	24 hour mode	1	Interrupt enable	1	Test Mode

**Table 4: Address Codes and Functions**

ADDRESS			FUNCTION					DATA					VALUE	
A4	A3	A2	A1	A0	HEX	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	00	0	0	0	0	0	0	0	0	0.00
0	0	0	0	1	01	0	0	0	0	0	0	0	0	0.25
0	0	0	1	0	02	0	0	0	0	0	0	0	0	1.12
0	0	0	1	1	03	0	0	0	0	0	0	0	0	1.50
0	0	1	0	0	04	0	0	0	0	0	0	0	0	0.50
0	0	1	0	1	05	0	0	0	0	0	0	0	0	0.75
0	0	1	1	0	06	0	0	0	0	0	0	0	0	1.25
0	0	1	1	1	07	0	0	0	0	0	0	0	0	1.75
0	1	0	0	0	08	M	M	M	M	M	M	M	M	0.00
0	1	0	0	1	09	M	M	M	M	M	M	M	M	0.25
0	1	0	1	0	0A	M	M	M	M	M	M	M	M	0.50
0	1	0	1	1	0B	M	M	M	M	M	M	M	M	0.75
0	1	1	0	0	0C	M	M	M	M	M	M	M	M	1.12
0	1	1	0	1	0D	M	M	M	M	M	M	M	M	1.50
0	1	1	1	0	0E	M	M	M	M	M	M	M	M	1.75
1	0	0	0	0	10	M	M	M	M	M	M	M	M	0.00
1	0	0	0	1	11	M	M	M	M	M	M	M	M	0.25

**NOTE:** Address 10010 to 11111 (12H to 1FH) are unused.  
 \* Unchecked bit for Interrupt Mask Register, M05 bit for Interrupt Status Register.  
 \* Indicates unused bit.  
 \* Indicates Mask bit is set (bit 4 of the Command Register).  
 \* Masked indicator bit is 12 hour format. Logic "0" indicates AM, logic "1" indicates PM.  
 \* M: Alarm compare for particular counter; M: Mask bit; M: Mask bit; M: Mask bit; M: Mask bit.

**Table 5: Interrupt and Status Registers Format**

INTERRUPT MASK REGISTER ADDRESS (100000, 10H) WRITE-ONLY					INTERRUPT STATUS REGISTER ADDRESS (10000B, 10H) READ-ONLY										
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
Bit	Pin	12	11	10	9	8	7	Bit	Pin	12	11	10	9	8	7

**Interrupt Operation**

The interrupt output N-channel MOSFET is active at all times when the Interrupt Enable bit is set (bit 4 of the Command Register), and operates in both the standby and battery backup modes.

Since system power is usually applied between VDD and VSS, the user can connect the Interrupt Source (pin #11) to VSS. This allows the Interrupt Output to turn on only while system power is applied and will not be pulled to VSS during standby operation. If interrupts are required only during standby operation, then the interrupt source pin should be connected to the battery's negative side (VBACKUP). In this configuration, for example, the interrupt could be used to turn on power for a cold boot.

**Power-Down Detector**

The ICM7170 contains an on-chip power-down detector that eliminates the need for external components for the battery back-up function. Whenever the voltage from the VBACKUP pin to the VSS pin is less than approximately 1.0VDC, the chip automatically switches to battery backup operation. Chip power is restored, operation is resumed to time counting and interrupt generation only. All other functions are disabled to achieve micropower standby power and to preserve time integrity.

If standby battery operation is not required the VBACKUP should be connected to VDD.

**APPLICATION NOTES**

**Time Synchronization**

Time synchronization is achieved through bit D0 of the Command Register, which is used to enable or disable the 100Hz clock from the counters. A logic "1" allows the counters to function and a logic "0" disables the counters. To accurately set the time, a logic "0" should be written into D0 and then the desired time entered into the appropriate counters. The clock is then started at the proper time by writing a logic "1" into D0 of the Command Register.

**Latched Data**

To prevent ambiguity while the processor is gathering data from the registers, the ICM7170 incorporates data latches and a transparent transition delay circuit.

By accessing the 100ths of seconds counter an internal store is generated and data from all the counters is stored into a 36-bit latch. A transition delay circuit will delay a 100Hz transition during a READ cycle until the internal store operation is completed. The data stored by the latches is then available for further processing until the 100ths of seconds counter is read again.

---

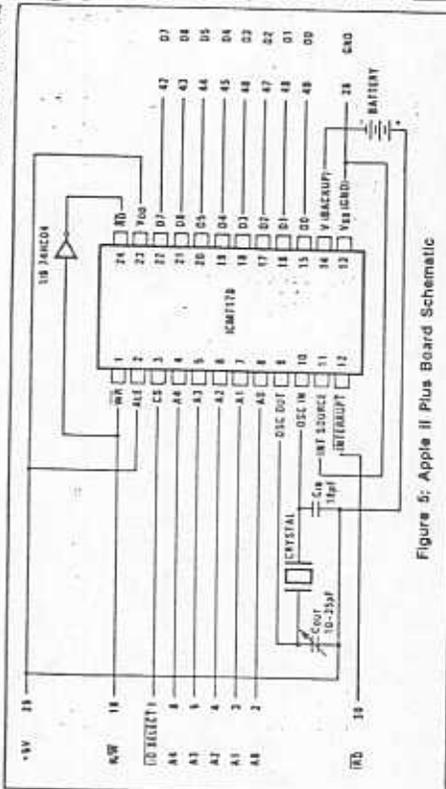


Figure 5: Apple II Plus Board Schematic

**Control Lines**

The RD, WR, and CS signals are active-low inputs. Data is placed on the bus from counters or registers when RD is a logic "0". Data is transferred to counters or registers when WR is a logic "0". RD and WR must be accompanied by a logical "0" CS as shown in Figures 3 and 4.

With the ALE (Address Latch Enable) input, the ICM7170 can be interfaced directly to microprocessors that use a multiplexed address/data bus by connecting the address lines A0-A4 to the data lines D0-D4. To address the chip, the address is placed on the bus and ALE is strobed. On the falling edge, the address and CS information is read into the address latch and buffer. RD and WR are used in the same way as on a non-multiplexed bus. If a non-multiplexed bus is used, ALE should be connected to VCC.

**Test Mode**

The test mode is entered by setting D5 of the Command Register to a logic "1". This connects the 100kHz pulse train to the seconds counter, and speeds up the counting functions.

**Oscillator Tuning**

Oscillator tuning should not be attempted by direct monitoring of the oscillator pins. Unless very specialized equipment is used, external connections to the oscillator pins cause capacitive loading of the crystal, and shift the oscillator frequency. As a result, the precision setting being attempted is corrupted. One indirect method of determining the oscillator frequency is to measure the period between interrupts on the Interrupt Output pin (#12). This measurement must be relative to the falling edges of the INTERRUPT pin. The oscillator setup and tuning can be performed as follows:

- 1) Select one of 4, readily-available oscillator frequencies and place the crystal between OSC IN (pin #10) and OSC OUT (pin #9).

**GENERAL**

The ICM7170 may clock such as U input. The cascaded output with the 8600 Baud rate.

Multi-Interrupt. This bit general. The 1-bit rate and low.

**ORDI**

- 2) Connect a load capacitor from OSC IN to Vcc.
- 3) Connect a variable capacitor from OSC OUT to Vcc. In cases where the crystal selected is a 32kHz Siatak type (C<sub>0</sub> = 9pF), the typical value of C<sub>0</sub> = 15pF and C<sub>OUT</sub> = 10-35pF.

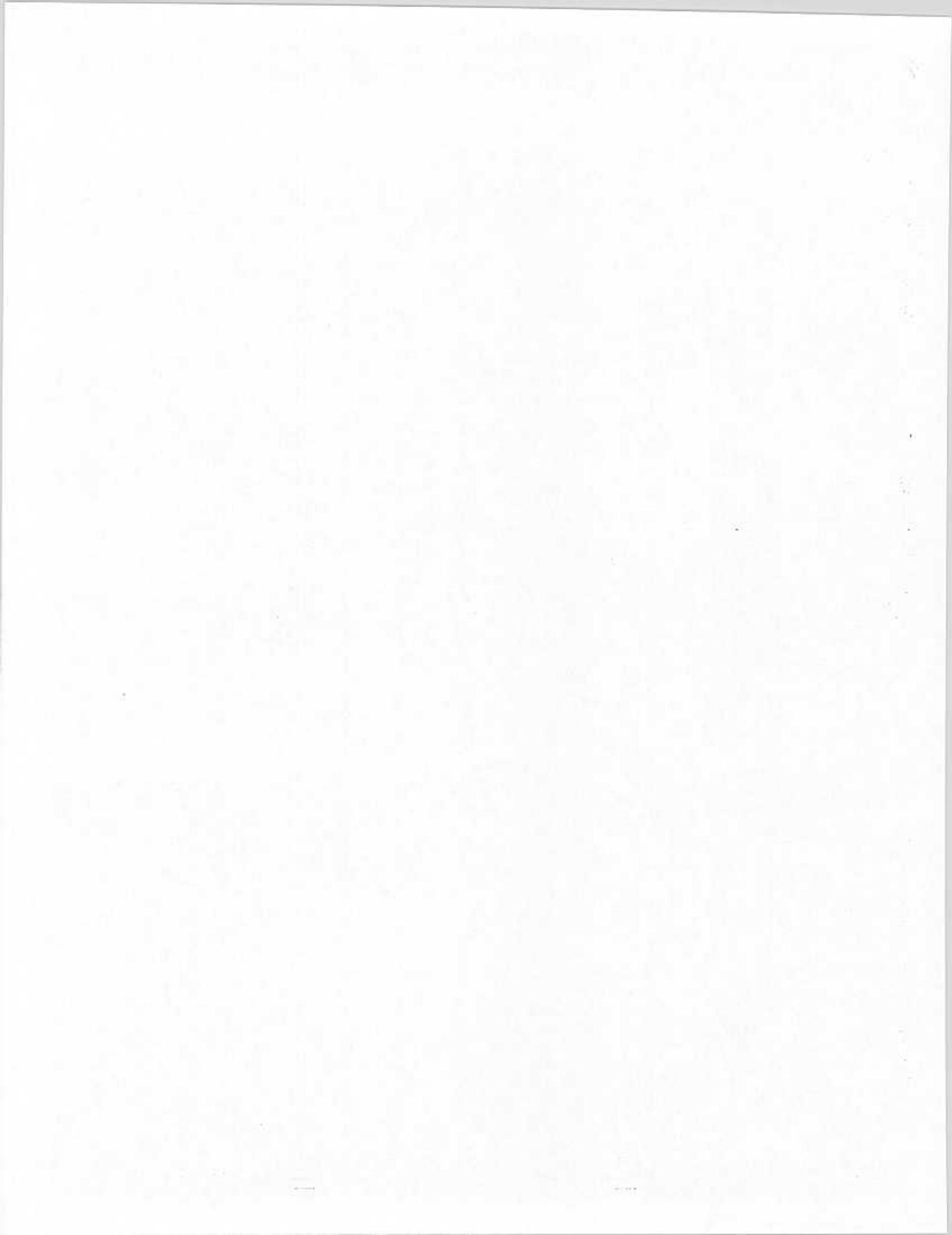
- 4) Place a 5KΩ resistor from the INTERRUPT pin to Vcc, and connect the INT SOURCE pin to Vcc standby mode.
- 5) Apply 5V power and insure the clock is not in standby mode.
- 6) Write all 0's to the Interrupt Mask Register, disabling all interrupts.
- 7) Write to the Command Register with the desired oscillator frequency, Hours mode (12 hour or 24 hour), Run = "1", Interrupt Enable = "1", and Test = "0".
- 8) Write to the Interrupt Mask Register, enabling one-second interrupts only.
- 9) Monitor the INTERRUPT output pin with a precision period counter and turn the OSC OUT capacitor for a reading of 1,000,000 seconds. The period counter must be triggered on the falling edge of the interrupt output for this measurement to be accurate.
- 10) Read the Interrupt Status Register. This action resets the interrupt output back to a logic "1" level.
- 11) Repeat steps 9 and 10 with a software loop. A suitable computer should be used.

**CIRCUIT APPLICATIONS**

**Apple II Plus Real-Time Clock**

Figure 5 shows the schematic of a board, using the ICM7170, that has been fabricated to plug into a slot in an Apple II Plus microcomputer. Very few external components are needed on the board to provide an interface to the Apple's 6502 MPU.

**80C88 MICROPROCESSOR**



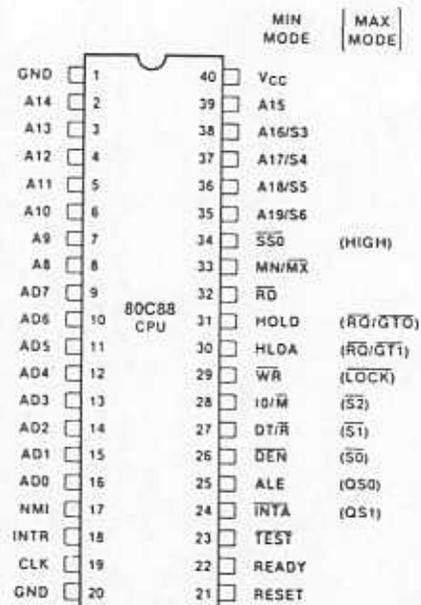
### Features

- COMPATIBLE WITH NMOS 8088
- DIRECT SOFTWARE COMPATIBILITY WITH 80C86, 8086, 8088
- 8 BIT DATA BUS INTERFACE
- 16 BIT INTERNAL ARCHITECTURE
- COMPLETELY STATIC DESIGN
  - ▶ DC - 5 MHz (80C88)
  - ▶ DC - 4 MHz (80C88-4)
- LOW POWER OPERATION
  - ▶ ICCSB = 500  $\mu$ A MAXIMUM
  - ▶ ICCOP = 10mA/MHz
- 1 MBYTE OF DIRECT MEMORY ADDRESSING CAPABILITY
- 24 OPERAND ADDRESSING MODES
- BIT, BYTE, WORD, AND BLOCK MOVE OPERATIONS
- 8 AND 16 BIT SIGNED/UNSIGNED ARITHMETIC
- BUS-HOLD CIRCUITRY ELIMINATES PULL-UP RESISTORS
- SCALED SAJI IV CMOS PROCESS
- SINGLE 5V POWER SUPPLY
- COMMERCIAL, INDUSTRIAL AND MILITARY TEMPERATURE RANGES

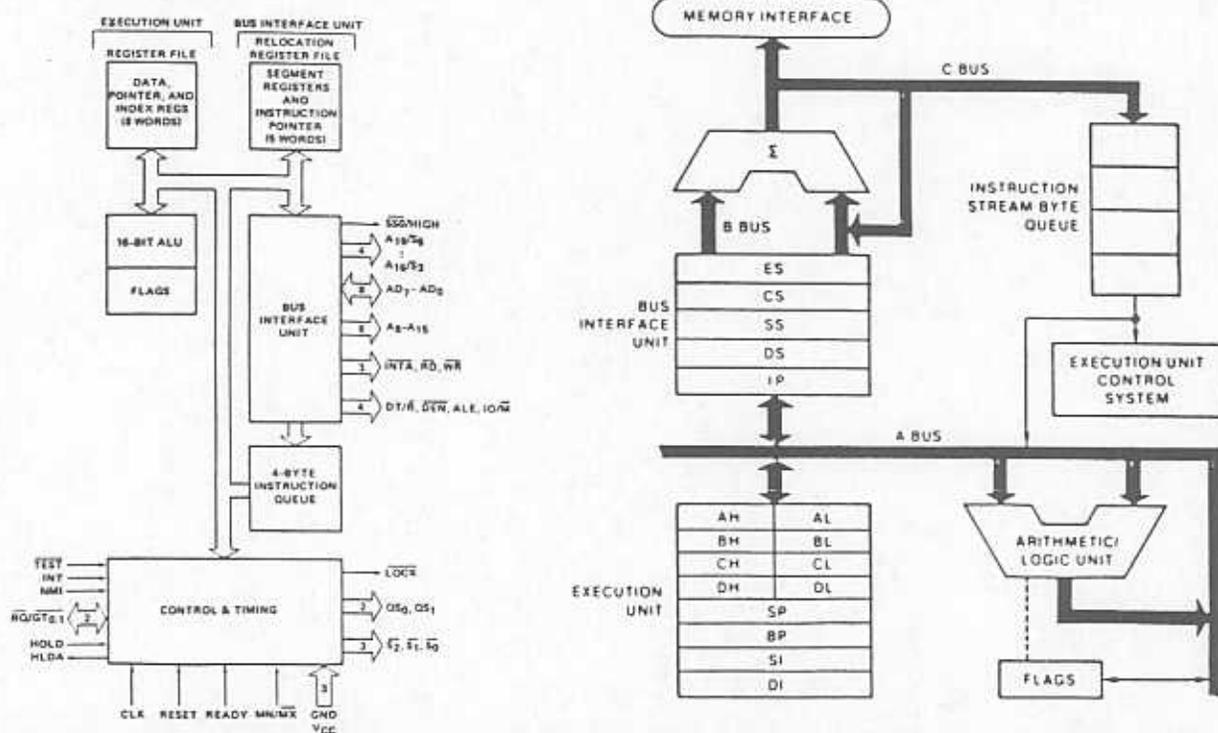
### Description

The Harris 80C88 high performance 8/16 bit CMOS CPU is manufactured using a self-aligned silicon gate CMOS process (Scaled SAJI IV). Two modes of operation, MINimum for small systems and MAXimum for larger applications such as multi-processing, allow user configuration to achieve the highest performance level. Full TTL compatibility and industry-standard operation allow use of existing NMOS 8088 hardware and Harris CMOS 80C86 peripherals. Complete software compatibility with the 80C86, 8086 and 8088 microprocessors allows use of existing software in new designs.

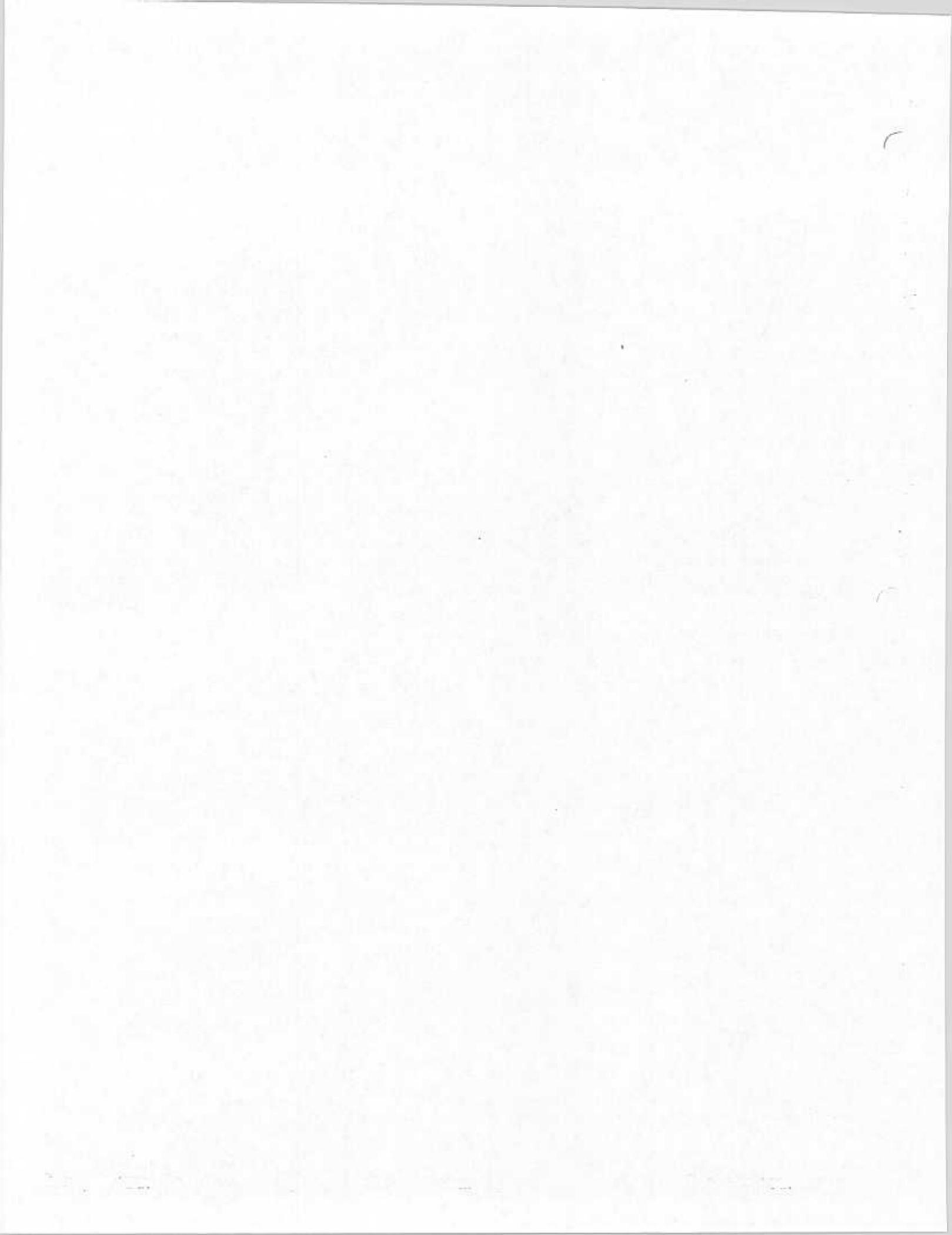
### Pinout



### Functional Diagram



CAUTION: Electronic devices are sensitive to electrostatic discharge. Proper I.C. handling procedures should be followed.



## Pin Description

The following pin function descriptions are for 80C88 systems in either minimum or maximum mode. The "local bus" in these

descriptions is the direct multiplexed bus interface connection to the 80C88 (without regard to additional bus buffers).

SYMBOL	PIN NUMBER	TYPE	NAME AND FUNCTION																		
AD7-AD0	9-16	I/O	Address Data Bus: These lines constitute the time multiplexed memory/I/O address (T1) and data (T2, T3, Tw, and T4) bus. These lines are active HIGH and are held at high impedance to the last valid logic level during interrupt acknowledge and local bus "hold acknowledge" or "grant sequence".																		
A15-A8	2-8, 39	O	Address Bus: These lines provide address bits 8 through 15 for the entire bus cycle (T1-T4). These lines do not have to be latched by ALE to remain valid. A15-A8 are active HIGH and are held at high impedance to the last valid logic level during interrupt acknowledge and local bus "hold acknowledge" or "grant sequence".																		
A19/S6, A18/S5, A17/S4, A16/S3	35 36 37 38	O O O O	<p>Address/Status: During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown.</p> <table border="1" data-bbox="1006 646 1421 814"> <thead> <tr> <th>S4</th> <th>S3</th> <th>CHARACTERISTICS</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>Alternate Data</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stack</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Code or None</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data</td> </tr> <tr> <td colspan="2">S6 is 0 (LOW)</td> <td></td> </tr> </tbody> </table> <p>This information indicates which segment register is presently being used for data accessing.</p> <p>These lines are held at high impedance to the last valid logic level during local bus "hold acknowledge" or "grant sequence".</p>	S4	S3	CHARACTERISTICS	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data	S6 is 0 (LOW)		
S4	S3	CHARACTERISTICS																			
0 (LOW)	0	Alternate Data																			
0	1	Stack																			
1 (HIGH)	0	Code or None																			
1	1	Data																			
S6 is 0 (LOW)																					
$\overline{RD}$	32	O	<p>Read: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the <math>\overline{IO/\overline{M}}</math> pin or S2. This signal is used to read devices which reside on the 80C88 local bus. <math>\overline{RD}</math> is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 80C88 local bus has floated.</p> <p>This line is held internally to a high impedance logic one state during "hold acknowledge" or "grant sequence".</p>																		
READY	22	I	READY: is the acknowledgment from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 82C84A clock generator to form READY. This signal is active HIGH. The 80C88 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.																		
INTR	18	I	Interrupt Request: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.																		
$\overline{TEST}$	23	I	$\overline{TEST}$ : input is examined by the "wait for test" instruction. If the $\overline{TEST}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.																		
NMI	17	I	NON-MASKABLE INTERRUPT: is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.																		
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must transition LOW to HIGH and remain active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.																		
CLK	19	I	Clock: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.																		
V <sub>CC</sub>	40		V <sub>CC</sub> : is the +5V $\pm$ 10% power supply pin.																		
GND	1, 20		GND: are the ground pins (Both pins must be connected to system ground).																		
MN/ $\overline{MX}$	33	I	Minimum/Maximum: indicates the mode in which the processor is to operate. The two modes are discussed in the following sections.																		

## Pin Description (continued)

The following pin descriptions are for the 80C88 system in maximum mode (i.e., MN/MX = GND). Only the pin functions

which are unique to maximum mode are described; all other pin functions are as described above.

SYMBOL	PIN NUMBER	TYPE	NAME AND FUNCTION																																				
$\overline{S2}, \overline{S1}, \overline{S0}$	26-28	O	<p>Status: is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 80C88 bus controller to generate all memory and I/O access control signals. Any change by <math>\overline{S2}</math>, <math>\overline{S1}</math>, or <math>\overline{S0}</math> during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or Tw is used to indicate the end of a bus cycle.</p> <p>These signals are held internally to a high impedance logic one state during "grant sequence".</p> <table border="1"> <thead> <tr> <th><math>\overline{S2}</math></th> <th><math>\overline{S1}</math></th> <th><math>\overline{S0}</math></th> <th>CHARACTERISTICS</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>0</td> <td>Interrupt Acknowledge</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>Read I/O port</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>Write I/O port</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Halt</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>0</td> <td>Code access</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Read memory</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Write memory</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>Passive</td> </tr> </tbody> </table>	$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	CHARACTERISTICS	0 (LOW)	0	0	Interrupt Acknowledge	0	0	1	Read I/O port	0	1	0	Write I/O port	0	1	1	Halt	1 (HIGH)	0	0	Code access	1	0	1	Read memory	1	1	0	Write memory	1	1	1	Passive
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	CHARACTERISTICS																																				
0 (LOW)	0	0	Interrupt Acknowledge																																				
0	0	1	Read I/O port																																				
0	1	0	Write I/O port																																				
0	1	1	Halt																																				
1 (HIGH)	0	0	Code access																																				
1	0	1	Read memory																																				
1	1	0	Write memory																																				
1	1	1	Passive																																				
$\overline{RQ/GT0}, \overline{RQ/GT1}$	30, 31	I/O	<p>Request/Grant: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with <math>\overline{RQ/GT0}</math> having higher priority than <math>\overline{RQ/GT1}</math>. <math>\overline{RQ/GT}</math> has internal bus-hold high circuitry and, if unused, may be left unconnected. The request/grant sequence is as follows (see Figure 5):</p> <ol style="list-style-type: none"> <li>1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 80C88 (pulse 1).</li> <li>2. During a T4 or T1 clock cycle, a pulse one clock wide from the 80C88 to the requesting master (pulse 2), indicates that the 80C88 has allowed the local bus to float and that it will enter the "grant sequence" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "grant sequence". The same rules as for HOLD/HOLDA apply as for when the bus is released.</li> <li>3. A pulse one CLK wide from the requesting master indicates to the 80C88 (pulse 3) that the "hold" request is about to end and that the 80C88 can reclaim the local bus at the next CLK. The CPU then enters T4.</li> </ol> <p>Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> <li>1. Request occurs on or before T2.</li> <li>2. Current cycle is not the low bit of a word.</li> <li>3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence.</li> <li>4. A locked instruction is not currently executing.</li> </ol> <p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> <li>1. Local bus will be released during the next clock.</li> <li>2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.</li> </ol>																																				
$\overline{LOCK}$	29	O	<p><math>\overline{LOCK}</math>: indicates that other system bus masters are not to gain control of the system bus while <math>\overline{LOCK}</math> is active (LOW). The <math>\overline{LOCK}</math> signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and is held high internally during "grant sequence".</p>																																				
QS1, QS0	24, 25	O	<p>Queue Status: provide status to allow external tracking of the internal 80C88 instruction queue.</p> <p>The queue status is valid during the CLK cycle after which the queue operation is performed. Note that the queue status never goes to a high impedance state (floated).</p> <table border="1"> <thead> <tr> <th>QS1</th> <th>QS0</th> <th>CHARACTERISTICS</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>No operation</td> </tr> <tr> <td>0</td> <td>1</td> <td>First byte of opcode from queue</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Empty the queue</td> </tr> <tr> <td>1</td> <td>1</td> <td>Subsequent byte from queue</td> </tr> </tbody> </table>	QS1	QS0	CHARACTERISTICS	0 (LOW)	0	No operation	0	1	First byte of opcode from queue	1 (HIGH)	0	Empty the queue	1	1	Subsequent byte from queue																					
QS1	QS0	CHARACTERISTICS																																					
0 (LOW)	0	No operation																																					
0	1	First byte of opcode from queue																																					
1 (HIGH)	0	Empty the queue																																					
1	1	Subsequent byte from queue																																					
--	34	O	<p>Pin 34 is always a logic one in the maximum mode and is internally held at a high impedance logic one during a "grant sequence".</p>																																				

# 80C88

## Pin Description

The following pin function descriptions are for the 80C88 minimum mode (i.e.,  $\overline{MN}/\overline{MX} = V_{CC}$ ). Only the pin functions which are unique to the minimum mode are described; all other pin functions are as described above.

SYMBOL	PIN NUMBER	TYPE	NAME AND FUNCTION
$\overline{IO}/\overline{M}$	28	O	Status Line: is an inverted maximum mode $\overline{S2}$ . It is used to distinguish a memory access from an I/O access. $\overline{IO}/\overline{M}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O = HIGH, M = LOW). $\overline{IO}/\overline{M}$ is held high impedance logic zero internally during local bus "hold acknowledge".
$\overline{WR}$	29	O	Write: strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the $\overline{IO}/\overline{M}$ signal. $\overline{WR}$ is active for T2, T3, and Tw of any write cycle. It is active LOW, and is held to high impedance logic one internally during local bus "hold acknowledge".
$\overline{INTA}$	24	O	INTA: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle. Note that $\overline{INTA}$ is never floated.
ALE	25	O	Address Latch Enable: is provided by the processor to latch the address into the 82C82/82C83 address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
$\overline{DT}/\overline{R}$	27	O	Data Transmit/Receive: is needed in a minimum system that desires to use an 82C86/82C87 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, $\overline{DT}/\overline{R}$ is equivalent to $\overline{S1}$ in the maximum mode, and its timing is the same as for $\overline{IO}/\overline{M}$ (T = HIGH, R = LOW). This signal is held to a high impedance logic one internally during local bus "hold acknowledge".
$\overline{DEN}$	26	O	Data Enable: is provided as an output enable for the 82C86/82C87 in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access, and for $\overline{INTA}$ cycles. For a read or $\overline{INTA}$ cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. $\overline{DEN}$ is held to high impedance logic one internally during local bus "hold acknowledge".
HOLD, HLDA	30 31	I O	HOLD: indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgment, in the middle of a T4 or T1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines.  Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.
$\overline{SS0}$	34	O	Status line: is logically equivalent to $\overline{S0}$ in the maximum mode. The combination of $\overline{SS0}$ , $\overline{IO}/\overline{M}$ , and $\overline{DT}/\overline{R}$ allows the system to completely decode the current bus cycle status. $\overline{SS0}$ is held to high impedance logic one during local bus "hold acknowledge".

$\overline{IO}/\overline{M}$	$\overline{DT}/\overline{R}$	$\overline{SS0}$	CHARACTERISTICS
1 (HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	Halt
0 (LOW)	0	0	Code access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

## Functional Description

### Static Operation

All 80C88 circuitry is static in design. Internal registers, counters and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on other microprocessors. The CMOS 80C88 can operate from DC to the appropriate upper frequency limit of 5 MHz. The processor clock may be stopped in either state (high/low) and held there indefinitely. This type of operation is especially useful for system debug or power critical applications.

The 80C88 can be single stepped using only the CPU clock. This state can be maintained as long as is necessary. Single step clock operation allows simple interface circuitry to provide critical information for start-up.

Static design also allows very low frequency operation (as low as DC). In a power critical situation, this can provide extremely low power operation since 80C88 power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until, at a DC input frequency, the power requirement is the 80C88 standby current.

### Internal Architecture

The internal functions of the 80C88 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the CPU block diagram.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 4 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 1 byte in the queue, the BIU will attempt a byte fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First-Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage.

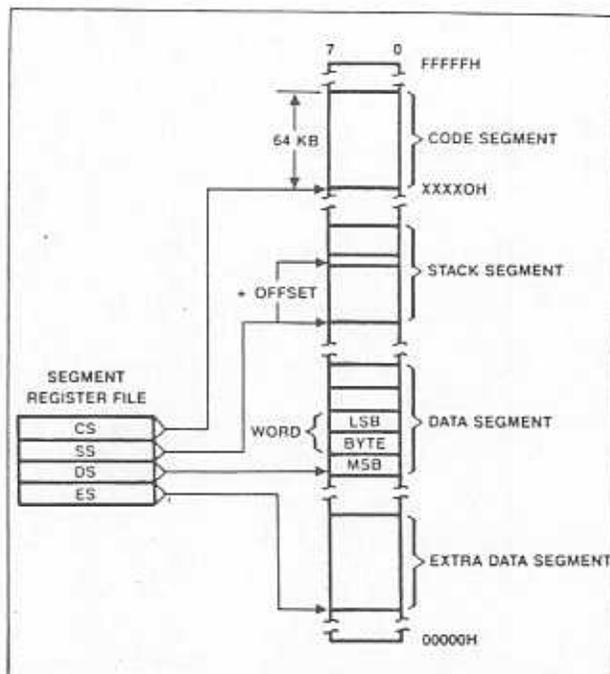


Figure 1. Memory Organization

### Memory Organization

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See FIGURE 1).

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g., code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations; Explicitly selected using a segment override.

Table 2.

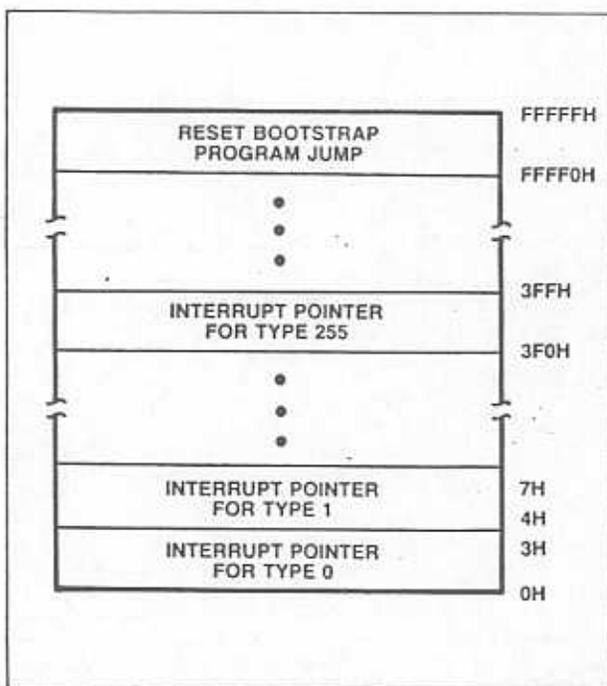


Figure 2. Reserved Memory Locations

The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See FIGURE 2). Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations

00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

#### Minimum and Maximum Modes

The requirements for supporting minimum and maximum 80C88 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 80C88 is equipped with a strap pin (MN/MX) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/MX pin is strapped to GND, the 80C88 defines pins 24 through 31 and 34 in maximum mode. When the MN/MX pin is strapped to VCC, the 80C88 generates bus control signals itself on pins 24 through 31 and 34.

The minimum mode 80C88 can be used with either a multiplexed or demultiplexed bus. This architecture provides the 80C88 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64K addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. An 82C86 or 82C87 transceiver can also be used if data bus buffering is required. (See FIGURE 3.) The 80C88 provides  $\overline{DEN}$  and  $DT/\overline{R}$  to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 82C88 bus controller (See FIGURE 4). The 82C88 decodes status lines  $S_0$ ,  $S_1$ , and  $S_2$ , and provides the system with all bus control signals. Moving the bus control to the 82C88 provides better source and sink current capability to the control lines, and frees the 80C88 pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 80C88 in maximum mode. These features allow co-processors in local bus and remote bus configurations.

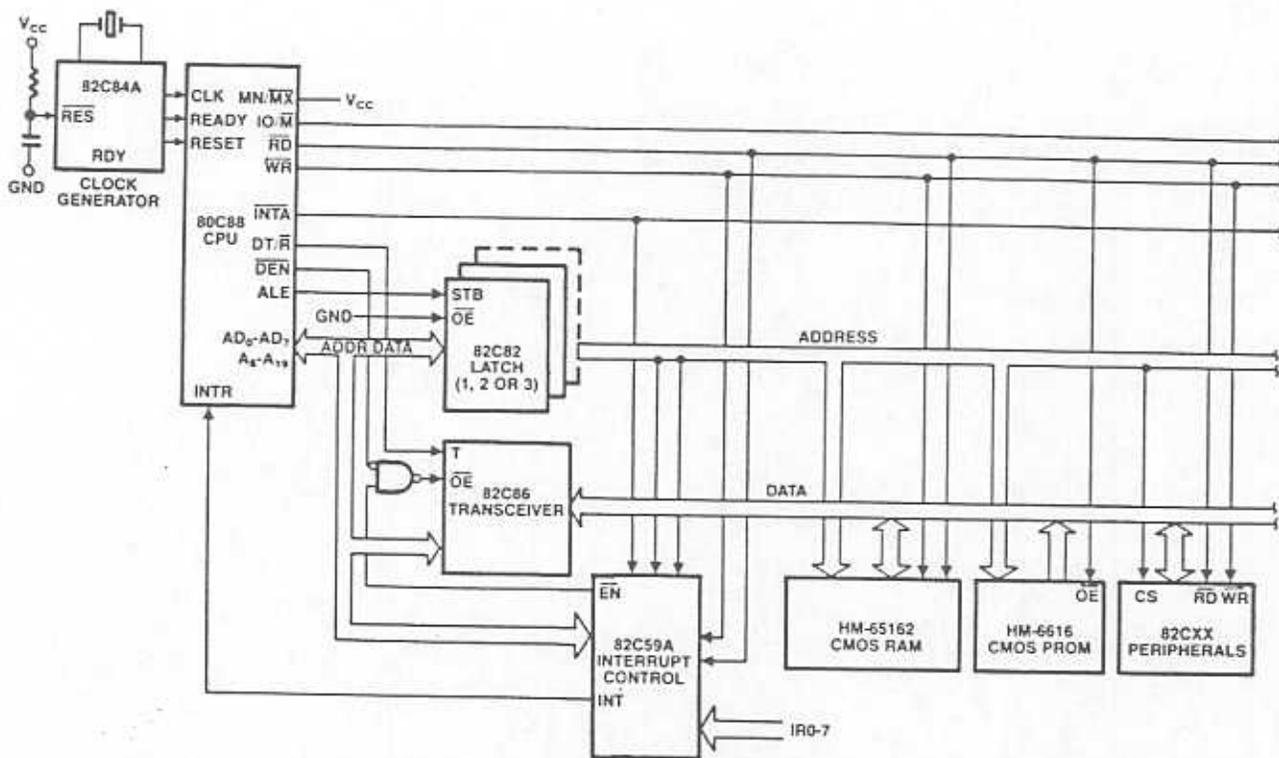


Figure 3. Demultiplexed Bus Configuration

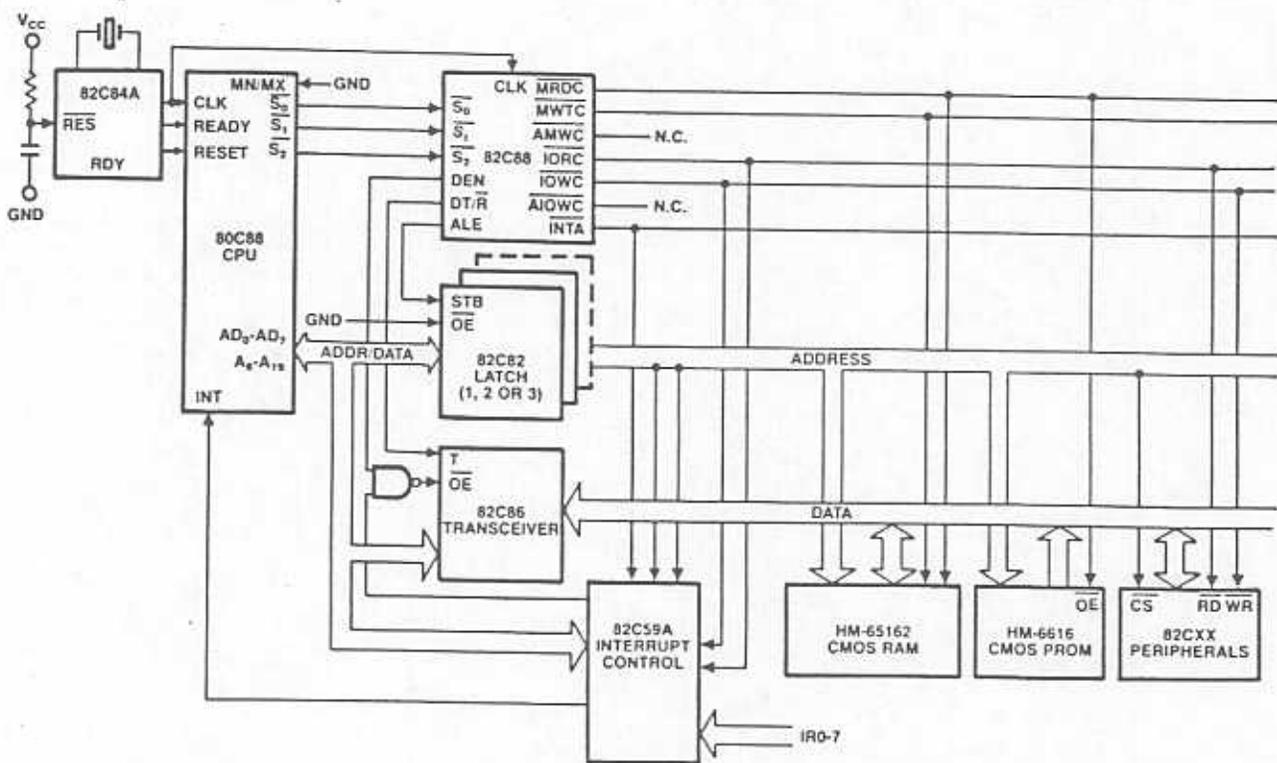


Figure 4. Fully Buffered System Using Bus Controller

### Bus Operation

The 80C88 address/data bus is broken into three parts – the lower eight address/data bits (AD0-AD7), the middle eight address bits (A8-A15), and the upper four address bits (A16-A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor, permitting the use of a standard 40 lead package. The middle eight address bits are not multiplexed, i.e. they remain valid throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4. (See FIGURE 5). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "wait" states (Tw) are inserted between T3 and T4. Each inserted "wait" state is of the same duration as a CLK cycle. Periods can occur between 80C88 driven bus cycles. These are referred to as "idle" states (Ti), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

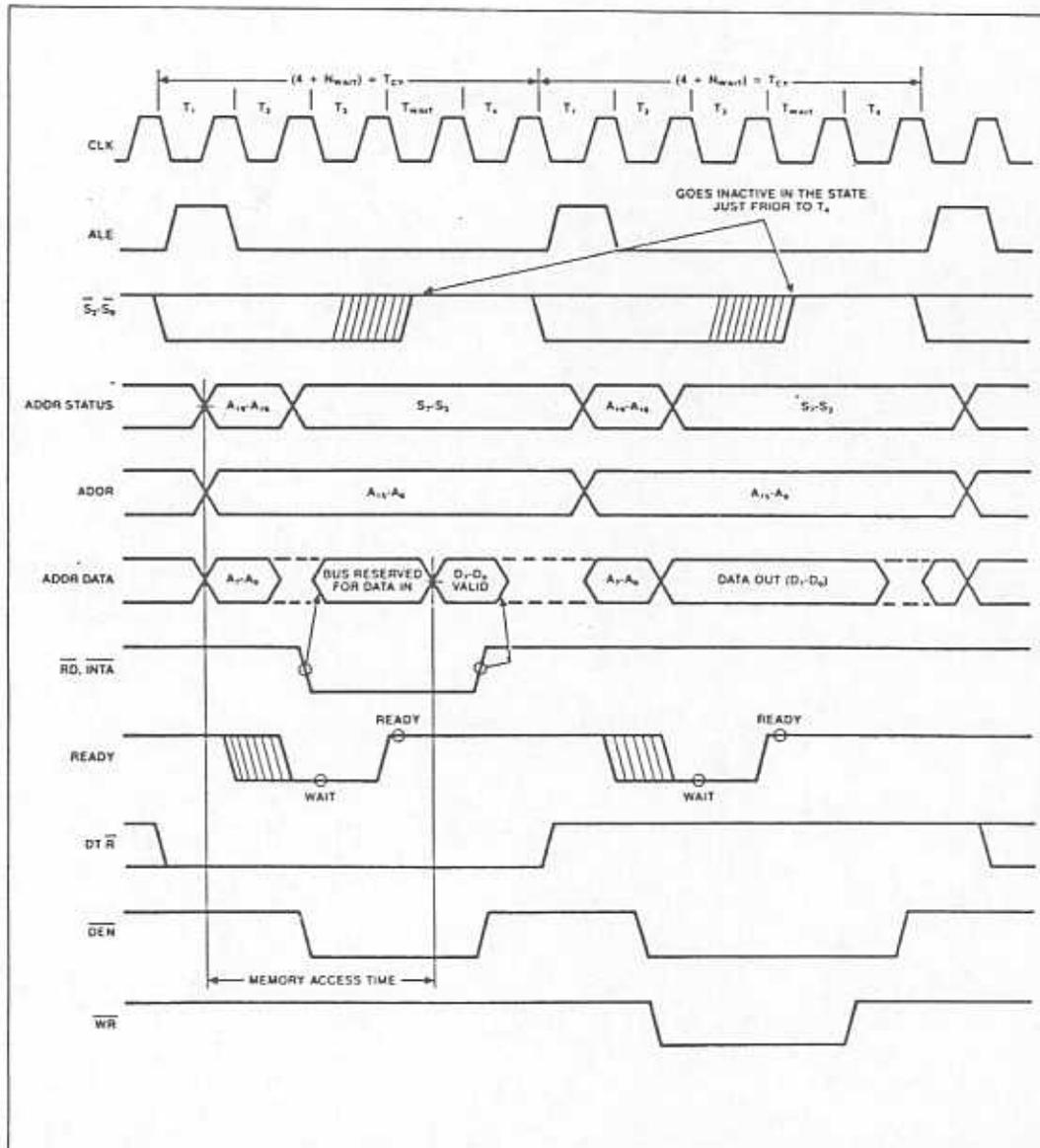


Figure 5. Basic System Timing

During T1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 82C88 bus controller, depending on the MN/MX strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits  $\overline{S_0}$ ,  $\overline{S_1}$ , and  $\overline{S_2}$  are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Table 3.

Status bits  $S_3$  through  $S_6$  are multiplexed with high order address bits and are therefore valid during T2 through T4.  $S_3$  and  $S_4$  indicate which segment register was used for this bus cycle in forming the address according to the following table:

$S_4$	$S_3$	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

Table 4.

$S_5$  is a reflection of the PSW interrupt enable bit.  $S_6$  is always equal to 0.

### I/O Addressing

In the 80C88, I/O operations can address up to a maximum of 64K I/O registers. The I/O address appears in the same format as the memory address on bus lines A15-A0. The address lines A19-A16 are zero in I/O operations. The variable I/O instructions, which use register DX as a pointer, have full address capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 address I/O with an 8-bit address on both halves of the 16-bit address bus. The 80C88 uses a full 16-bit address on its lower 16 address lines.

### External Interface

#### Processor Reset and Initialization

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 80C88 RESET is required to be HIGH for greater than four clock cycles. The 80C88 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 80C88 operates normally, beginning with the instruction in absolute location FFFF0H (see FIGURE 2). The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50  $\mu$ s after power up, to allow complete initialization of the 80C88.

If INTR is asserted sooner than nine clock cycles after the end of RESET, the processor may execute one instruction before responding to the interrupt.

#### Bus Hold Circuitry

To avoid high current conditions caused by floating inputs to CMOS devices and to eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on 80C88 pins 2-16, 26-32 and 34-39 (see FIGURE 6A, 6B). These circuits maintain a valid logic state if no driving source is present (i.e.,

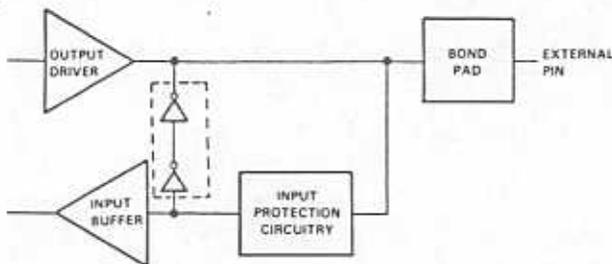


Figure 6A. Bus hold circuitry pin 2-16, 35-39.

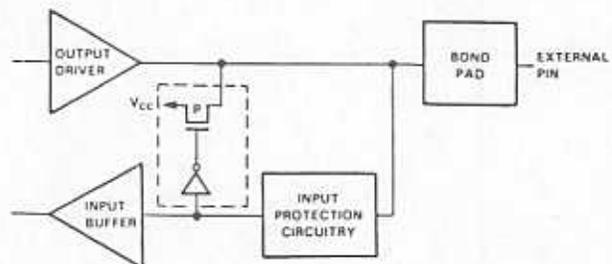


Figure 6B. Bus hold circuitry pin 26-32, 34.

an unconnected pin or a driving source which goes to a high impedance state).

To overdrive the "bus hold" circuits, an external driver must be capable of supplying 400 $\mu$ A minimum sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a "resistive" type element, the associated power supply current is negligible. Power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

### Interrupt Operations

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see FIGURE 2), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

### Non-Maskable Interrupt (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any high going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may

occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure.

The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

### Maskable Interrupt (INTR)

The 80C88 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK.

To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. INTR may be removed anytime after the falling edge of the first INTA signal. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step). The FLAGS register, which is automatically pushed onto the stack, reflects the state of the processor prior to the interrupt. The enable bit will be zero until the old FLAGS register is restored, unless specifically set by an instruction.

During the response sequence (see FIGURE 7), the processor executes two successive (back to back) interrupt acknowledge cycles. The 80C88 emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a byte is fetched from the external interrupt system (e.g., 82C59A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table.

An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. INTR may be removed anytime after the falling edge of the first INTA signal. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

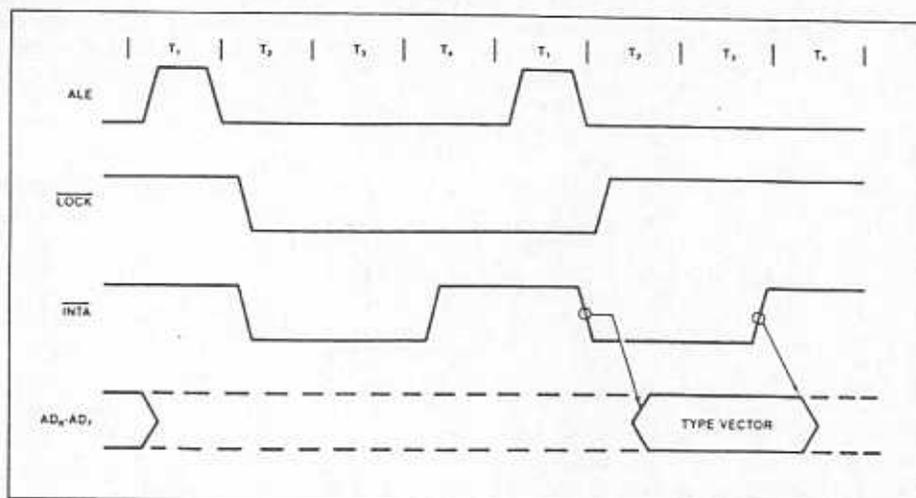


Figure 7. Interrupt Acknowledge Sequence

## Halt

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on IO/M, DT/R, and SS0. In maximum mode, the processor issues appropriate HALT status on S2, S1, and S0, and the 82C88 bus controller issues one ALE. The 80C88 will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 80C88 out of the HALT state.

## Read/Modify/Write (Semaphore) Operations Via LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The LOCK signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While LOCK is active, a request on a RQ/GT pin will be recorded, and then honored at the end of the LOCK.

## External Synchronization Via TEST

As an alternative to interrupts, the 80C88 provides a single software-testable input pin (TEST). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the TEST input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 80C88 3-states all output drivers while inputs and I/O pins are held at valid logic levels by internal bus-hold circuits. If interrupts are enabled, the 80C88 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

## Basic System Timing

In minimum mode, the MN/MX pin is strapped to V<sub>CC</sub> and the processor emits bus control signals (RD, WR, IO/M, etc.) directly. In maximum mode, the MN/MX pin is strapped to GND and the processor emits coded status information which the 82C88 bus controller uses to generate MULTIBUS™ compatible bus control signals.

## System Timing – Minimum System

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal (See FIGURE 5). The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0-AD7) at this time, into the 82C82/82C83 latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the IO/M signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus is held at the last valid logic state by internal bus-hold devices. The read control signal is also asserted at T2. The read (RD) signal causes the addressed device to enable its data bus drivers to

the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (82C86/82C87) is required to buffer the local bus, signals DT/R and DEN are provided by the 80C88.

A write cycle also begins with the assertion of ALE and the emission of the address. The IO/M signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and Tw, the processor asserts the write control signal. The write (WR) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for output drivers to become inactive.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge (INTA) signal is asserted in place of the read (RD) signal and the address bus is held at the last valid logic state by internal bus-hold devices (see FIGURE 6). In the second of two successive INTA cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e., 82C59A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

## Bus Timing – Medium Complexity Systems

For medium complexity systems, the MN/MX pin is connected to GND and the 82C88 bus controller is added to the system, as well as an 82C82/82C83 latch for latching the system address, and an 82C86/82C87 transceiver to allow for bus loading greater than the 80C88 is capable of handling (see FIGURE 8). Signals ALE, DEN, and DT/R are generated by the 82C88 instead of the processor in this configuration, although their timing remains relatively the same. The 80C88 status outputs (S2, S1, and S0) provide type of cycle information and become 82C88 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 82C88 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 82C88 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The 82C86/82C87 transceiver receives the usual T and OE inputs from the 82C88 DT/R and DEN outputs.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 82C59A located on either the local bus or the system bus. If the master 82C59A priority interrupt controller is positioned on the local bus, the 82C86/82C87 transceiver must be disabled when reading from the master 82C59A during the interrupt acknowledge sequence and software "poll".

## The 80C88 Compared To The 80C86

The 80C88 CPU is an 8-bit processor designed around the 8086 internal structure. Most internal functions of the 80C88 are identical to the equivalent 80C86 functions. The 80C88 handles the external bus the same way the 80C86 does with the distinction of handling only 8 bits at a time. Sixteen-bit

MULTIBUS™ is a trademark of Intel Corp.

operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. Internally, there are three differences between the 80C88 and the 80C86. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80C88, whereas the 80C86 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80C88 BIU will fetch a new instruction to load into the queue each time there is a 1 byte space available in the queue. The 80C86 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 80C88 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80C88 and 80C86 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 80C88 or an 80C86.

The hardware interface of the 80C88 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

- A8-A15 – These pins are only address outputs on the 80C88. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{\text{BHE}}$  has no meaning on the 80C88 and has been eliminated.
- $\overline{\text{SS0}}$  provides the  $\overline{\text{S0}}$  status information in the minimum mode. This output occurs on pin 34 in minimum mode only.  $\text{DT}/\overline{\text{R}}$ ,  $\text{IO}/\overline{\text{M}}$ , and  $\overline{\text{SS0}}$  provide the complete bus status in minimum mode.
- $\text{IO}/\overline{\text{M}}$  has been inverted to be compatible with the 8085 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

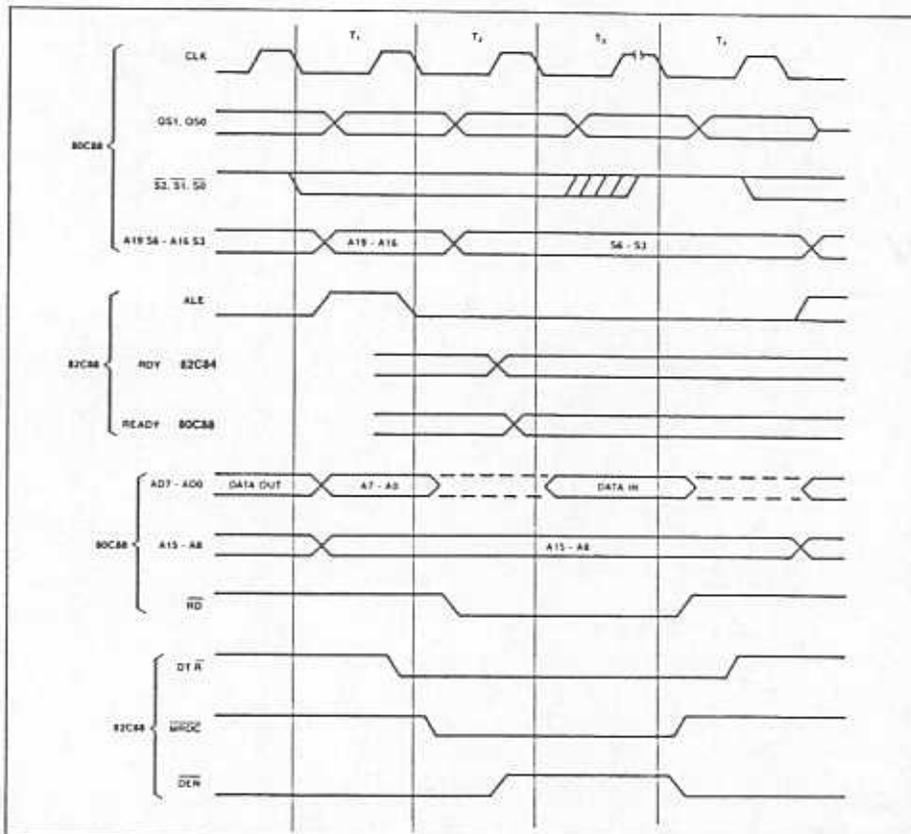


Figure 8. Medium Complexity System Timing

## Specifications

## ABSOLUTE MAXIMUM RATINGS

Supply Voltage	+8.0 Volts	Operating Temperature Range	
Operating Voltage Range	+4V to +7V	Commercial	0°C to +70°C
Input Voltage Applied	GND - 2.0V to 6.5V	Industrial	-40°C to +85°C
Output or I/O Voltage Applied	GND - 0.5V to VCC + 0.5V	Military	-55°C to +125°C
Storage Temperature Range	-65°C to +150°C	Maximum Package Power Dissipation	1 Watt

**CAUTION:** Stresses above those listed in the "ABSOLUTE MAXIMUM RATINGS" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

## D.C. ELECTRICAL CHARACTERISTICS

VCC = 5.0V ± 10%; T<sub>A</sub> = 0°C to +70°C (C80C88); T<sub>A</sub> = -40°C to +85°C (I80C88); T<sub>A</sub> = -55°C to +125°C (M80C88)

SYMBOL	PARAMETER	MIN	MAX	UNITS	TEST CONDITIONS
VIH	Logical One Input Voltage	2.0		V	C80C88, I80C88
		2.2		V	M80C88
VIL	Logical Zero Input Voltage		0.8	V	
VIHC	CLK Logical One Input Voltage	VCC - 0.8V		V	
VILC	CLK Logical Zero Input Voltage		0.8	V	
VOH	Output High Voltage	3.0		V	IOH = -2.5mA
		VCC - 0.4		V	IOH = -100µA
VOL	Output Low Voltage		0.4	V	IOL = +2.5mA
IIL	Input Leakage Current	-1.0	1.0	µA	OV ≤ VIN ≤ VCC
IBHH	Input Current - Bus Hold High	-40	-400	µA	VIN = 3.0V (see Note 1)
IBHL	Input Current - Bus Hold Low	40	400	µA	VIN = 0.8V (see Note 2)
IO	Output Leakage Current	-10.0	10.0	µA	OV ≤ VO ≤ VCC
ICCSB	Standby Power Supply Current		500	µA	VCC = 5.5V See Note 3.
ICCOP	Operating Power Supply Current		10	mA/MHz	VCC = 5.5V

## CAPACITANCE

T<sub>A</sub> = 25°C; VCC = GND = 0V; VIN = +5V or GND

SYMBOL	PARAMETER	MIN	MAX	UNITS	TEST CONDITIONS
CIN*	Input Capacitance		5	pf	FREQ = 1MHz Unmeasured pins returned to GND
COUT*	Output Capacitance		15	pf	
CIO*	I/O Capacitance		20	pf	

\* Guaranteed and sampled, but not 100% tested

Note 1: IBHH should be measured after raising VIN to VCC and then lowering to 3.0V on the following pins: 2-16, 26-32, 34-39.

Note 2: IBHL should be measured after lowering VIN to GND and then raising to 0.8V on the following pins: 2-16, 35-39.

Note 3: ICCSB tested during clock high time after HALT instruction execution. VIN = VCC or GND. VCC = 5.5V. Outputs unloaded.

## Specifications (continued)

## A.C. ELECTRICAL CHARACTERISTICS

VCC = 5V ± 10%; (C80C88: T<sub>A</sub> = 0°C to +70°C)  
 (I80C88: T<sub>A</sub> = -40°C to +85°C)  
 (M80C88: T<sub>A</sub> = -55°C to +125°C)

## MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

SYMBOL	PARAMETER	80C88-4		80C88		UNITS	TEST CONDITIONS
		MIN	MAX	MIN	MAX		
TCLCL	CLK Cycle Period	250		200		ns	
TCLCH	CLK Low Time	151		118		ns	
TCHCL	CLK High Time	85		69		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		30		ns	
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84A (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84A (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C88	118		118		ns	
TCHRYX	READY Hold Time into 80C88	30		30		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	-8		-8		ns	
THVCH	HOLD Setup Time	35		35		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		30		ns	
TILIH	Input Rise Time (Except CLK)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		15		15	ns	From 2.0V to 0.8V

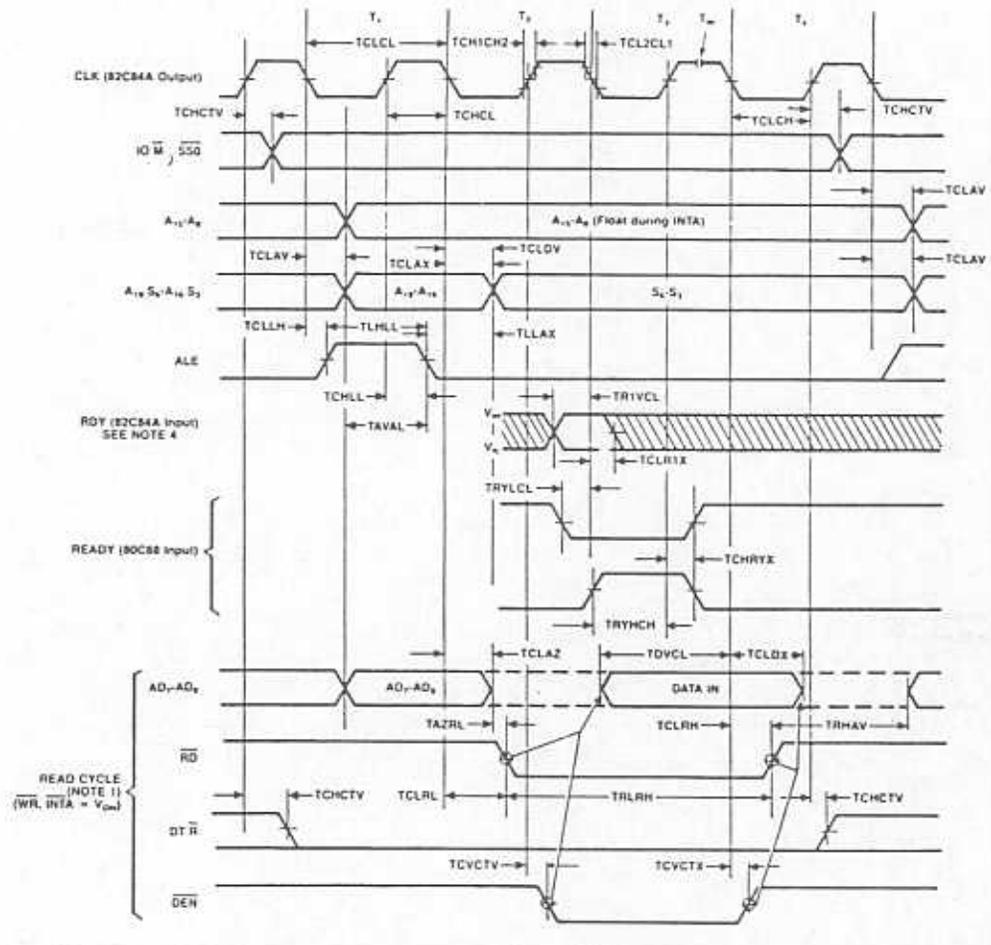
A.C. ELECTRICAL CHARACTERISTICS (Continued)  
 MINIMUM COMPLEXITY SYSTEM TIMING RESPONSES

SYMBOL	PARAMETER	80C88-4		80C88		UNITS	TEST CONDITIONS
		MIN	MAX	MIN	MAX		
TCLAV	Address Valid Delay	10	110	10	110	ns	C <sub>L</sub> = 100 pF for all 80C88 Outputs in addition to internal loads
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	80	ns	
TCHSZ	Status Float Delay		80		80	ns	
TLHLL	ALE Width	TCLCH-20		TCLCH-20		ns	
TCLLH	ALE Active Delay		80		80	ns	
TCHLL	ALE Inactive Delay		85		85	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL-10		TCHCL-10		ns	
TCLDV	Data Valid Delay	10	110	10	110	ns	
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time After $\overline{WR}$	TCLCH-30		TCLCH-30		ns	
TCVCTV	Control Active Delay 1	10	110	10	110	ns	
TCHCTV	Control Active Delay 2	10	110	10	110	ns	
TCVCTX	Control Inactive Delay	10	110	10	110	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	165	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	150	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL-45		TCLCL-45		ns	
TCLHAV	HLDA Valid Delay	10	160	10	160	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL-75		2TCLCL-75		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL-60		2TCLCL-60		ns	
TAVAL	Address Valid to ALE Low	TCLCH-60		TCLCH-60		ns	
TOLOH	Output Rise Time		15		15	ns	From 0.8V to 2.0V From 2.0V to 0.8V
TOHOL	Output Fall Time		15		15	ns	

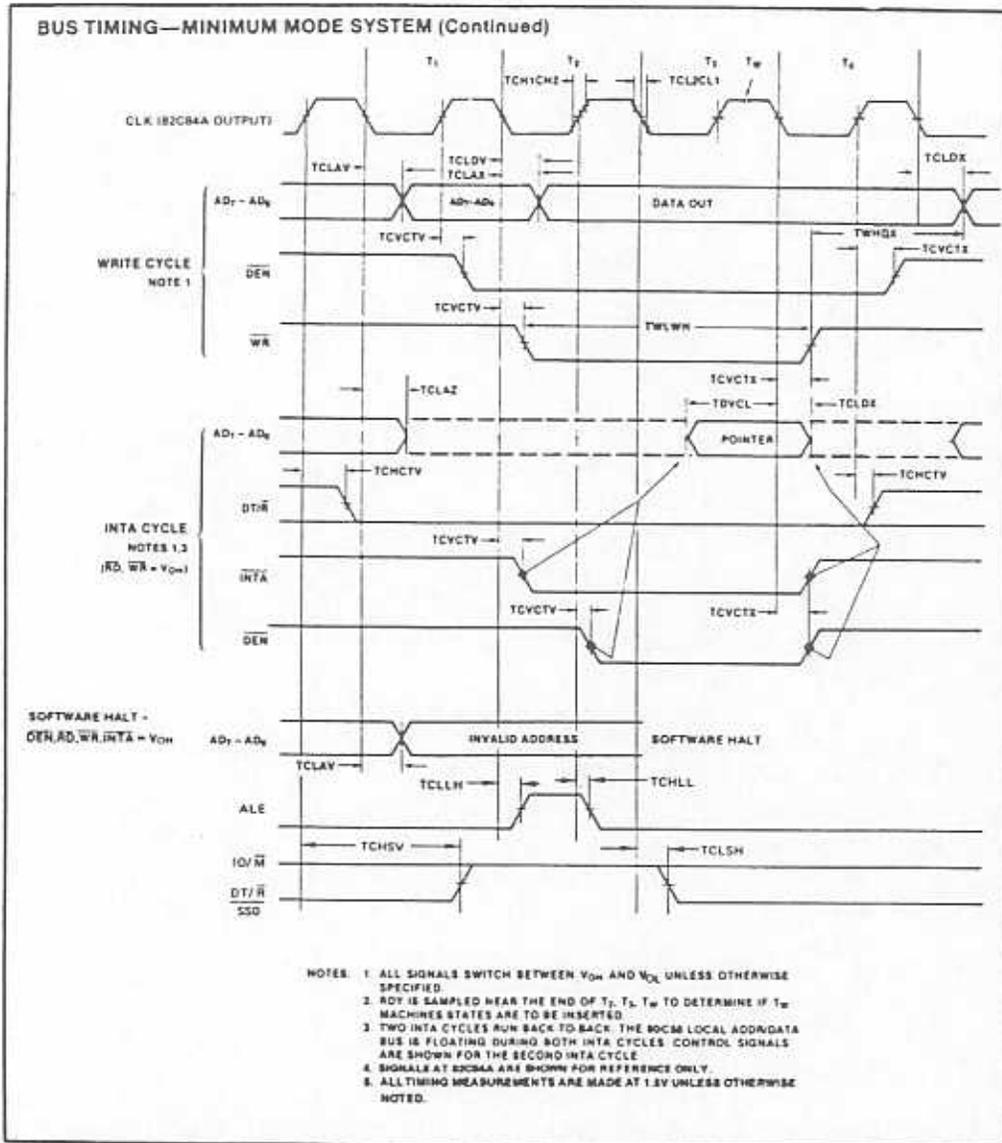
- NOTES: 1. Signal at 82C84A shown for reference only.  
 2. Setup requirement for asynchronous signal only to guarantee recognition at next clock.  
 3. Applies only to T2 state (8 nanoseconds into T3).

# Waveforms

## BUS TIMING - MINIMUM MODE SYSTEM



## Waveforms (Continued)



## 80C88

A.C. ELECTRICAL CHARACTERISTICS (Continued)  
 MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER) TIMING REQUIREMENTS

SYMBOL	PARAMETER	80C88-4		80C88		UNITS	TEST CONDITIONS
		MIN	MAX	MIN	MAX		
TCLCL	CLK Cycle Period	250		200		ns	
TCLCH	CLK Low Time	151		118		ns	
TCHCL	CLK High Time	85		69		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		30		ns	
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C88	118		118		ns	
TCHRYX	READY Hold Time into 80C88	30		30		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	-8		-8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	30		30		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	30		30		ns	
TCHGX	$\overline{RQ}$ Hold Time into 80C88 (See Note 4)	40	TCHCL	40	TCHCL	ns	
TILIH	Input Rise Time (Except CLK)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		15		15	ns	From 2.0V to 0.8V

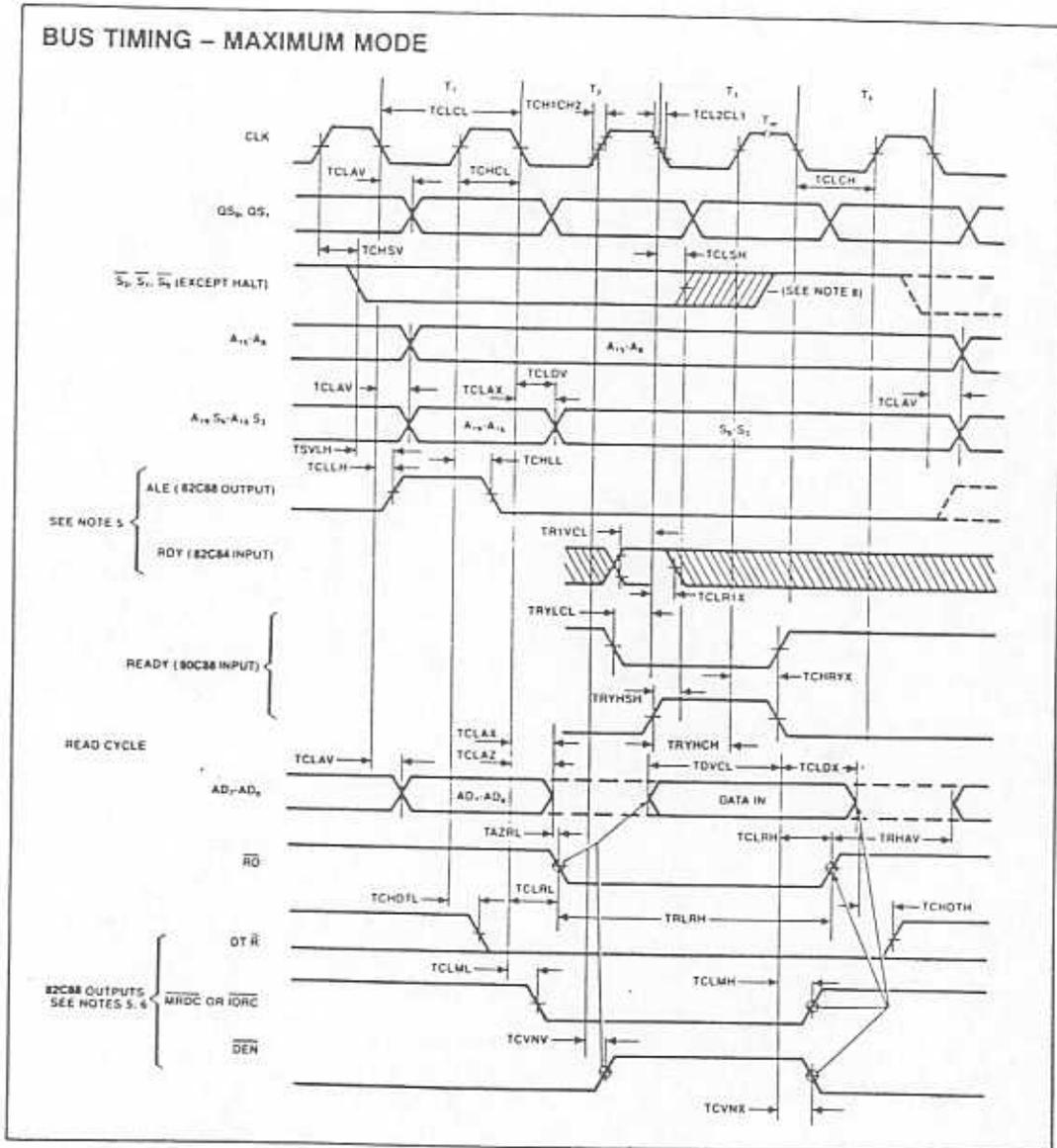
## A.C. CHARACTERISTICS

## MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER) TIMING RESPONSES

SYMBOL	PARAMETER	80C88-4		80C88		UNITS	TEST CONDITIONS	
		MIN	MAX	MIN	MAX			
TCLML	Command Active Delay (See Note 1)	5	35	5	35	ns	C <sub>L</sub> = 100 pF for all 80C88 Outputs in addition to internal loads	
TCLMH	Command Inactive Delay (See Note 1)	5	35	5	35	ns		
TRYHSH	READY Active to Status Passive (See Notes 3, 5)		110		110	ns		
TCHSV	Status Active Delay	10	110	10	110	ns		
TCLSH	Status Inactive Delay (See Note 5)	10	130	10	130	ns		
TCLAV	Address Valid Delay	10	110	10	110	ns		
TCLAX	Address Hold Time	10		10		ns		
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	80	ns		
TCHSZ	Status Float Delay		80		80	ns		
TSVLH	Status Valid to ALE High (See Note 1)		20		20	ns		
TSVMCH	Status Valid to MCE High (See Note 1)		30		30	ns		
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15	ns		
TCLMCH	CLK Low to MCE High (See Note 1)		25		25	ns		
TCHLL	ALE Inactive Delay (See Note 1)	4	18	4	18	ns		
TCLMCL	MCE Inactive Delay (See Note 1)		15		15	ns		
TCLDV	Data Valid Delay	10	110	10	110	ns		
TCHDX	Data Hold Time	10		10		ns		
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	ns		
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	ns		
TAZRL	Address Float to Read Active	0		0		ns		
TCLRL	$\overline{RD}$ Active Delay	10	165	10	165	ns		
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	150	ns		
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL-45		TCLCL-45		ns		
TCHDTL	Direction Control Active Delay (See Note 1)		50		50	ns		
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30	ns		
TCLGL	$\overline{GT}$ Active Delay		85		85	ns		
TCLGH	$\overline{GT}$ Inactive Delay		85		85	ns		
TRLRH	$\overline{RD}$ Width	2TCLCL-75		2TCLCL-75		ns		
TOLOH	Output Rise Time		15		15	ns		From 0.8V to 2.0V
TOHOL	Output Fall Time		15		15	ns		From 2.0V to 0.8V

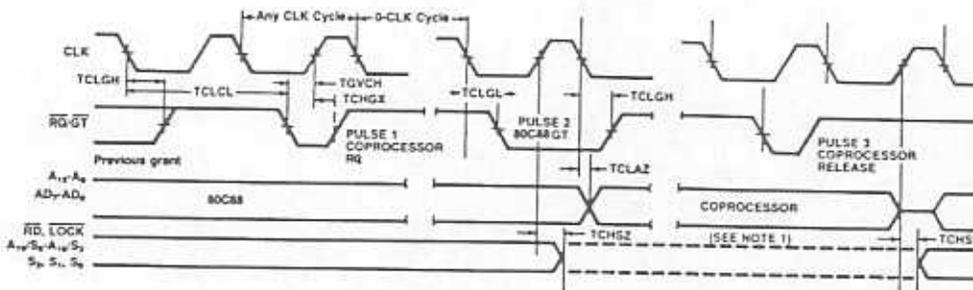
- NOTES: 1. Signal at 82C84A shown for reference only.  
 2. Setup requirement for asynchronous signal only to guarantee recognition at next clock.  
 3. Applies only to T2 state (8 nanoseconds into T3).  
 4. The 80C88 actively pulls the  $\overline{RD}/\overline{GT}$  pin to a logic one on the following clock low time.  
 5. Status lines return to their inactive (logic one) state after CLK goes low and READY goes high.

Waveforms



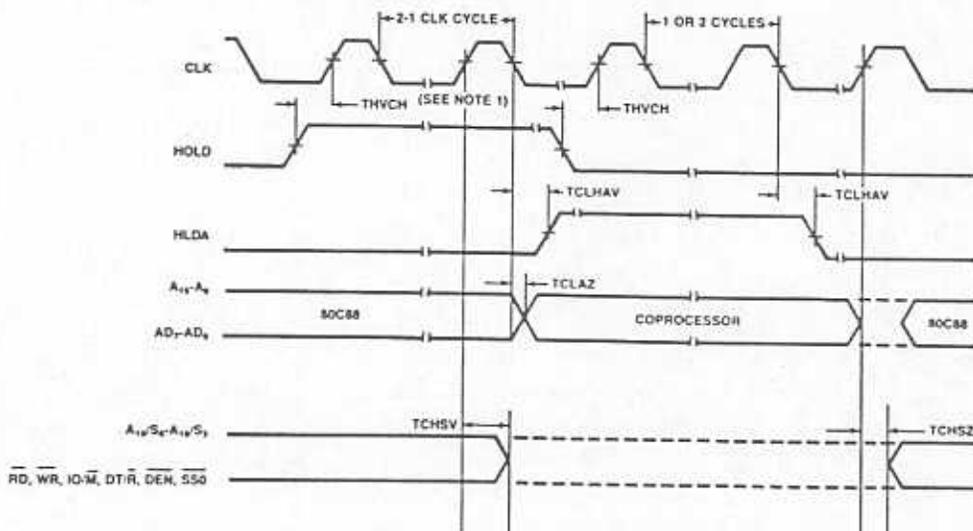


REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)

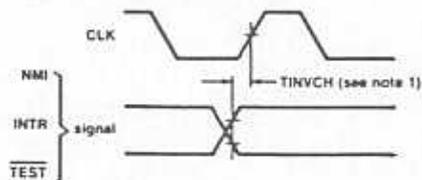


NOTE: 1. THE COPROCESSOR MAY NOT DRIVE THE BUSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)

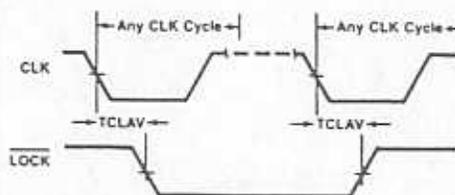


ASYNCHRONOUS SIGNAL RECOGNITION

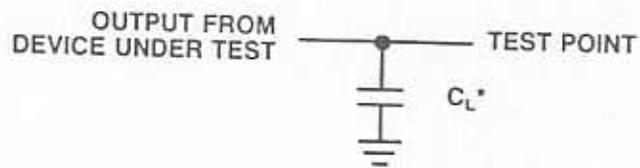


NOTE: 1. SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK.

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)

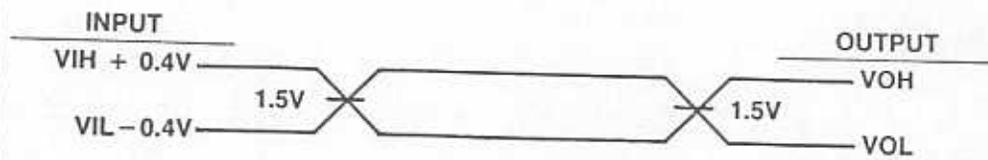


## A.C. Test Circuits



\*Includes stray and jig capacitance

## A.C. Testing Input, Output Waveform



A.C. Testing: All input signals (other than CLK) must switch between  $V_{IL_{max}} - 0.4V$  and  $V_{IH_{min}} + 0.4V$ .  
 CLK must switch between  $0.4V$  and  $V_{CC} - 0.4V$ .  $T_R$  and  $T_F$  must be less than or equal to 15ns.  
 CLK  $T_R$  and  $T_F$  must be less than or equal to 10ns.

Instruction Set Summary

**DATA TRANSFER**

**MOV - Move**

Register/memory to/from register	1000100w	mod reg r/m		
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w = 1
Immediate to register	1011w	reg	data	data if w = 1
Memory to accumulator	1010000w	addr low	addr high	
Accumulator to memory	1010001w	addr low	addr high	
Register/memory to segment register	10001110	mod 0 reg r/m		
Segment register to register/memory	10001100	mod 0 reg r/m		

**PUSH - Push**

Register/memory	11111111	mod 110 r/m		
Register	01010	reg		
Segment register	000	reg 1-6		

**POP - Pop**

Register/memory	10001111	mod 000 r/m		
Register	01011	reg		
Segment register	000	reg 1-6		

**EXCH - Exchange**

Register/memory with register	1000011w	mod reg r/m		
Register with accumulator	10010	reg		

**IN - Input from**

Fixed port	1110010w	port		
Variable port	1110110w			

**OUT - Output to**

Fixed port	1110011w	port		
Variable port	1110111w			

**XLAT - Translate byte to AL**

	11010111			
--	----------	--	--	--

**LEA - Load EA to register**

	10001101	mod reg r/m		
--	----------	-------------	--	--

**LDS - Load pointer to DS**

	11000101	mod reg r/m		
--	----------	-------------	--	--

**LDS - Load pointer to ES**

	11000100	mod reg r/m		
--	----------	-------------	--	--

**LAMF - Load AH with flags**

	10011111			
--	----------	--	--	--

**SAMF - Store AH into flags**

	10011110			
--	----------	--	--	--

**PUSHF - Push flags**

	10011100			
--	----------	--	--	--

**POPF - Pop flags**

	10011101			
--	----------	--	--	--

**ARITHMETIC**

**ADD - Add**

Reg. memory with register to either	0000000w	mod reg r/m		
Immediate to register/memory	1000001w	mod 000 r/m	data	data if w = 1
Immediate to accumulator	0000010w	data	data if w = 1	

**ADC - Add with carry**

Reg. memory with register to either	0001000w	mod reg r/m		
Immediate to register/memory	1000001w	mod 010 r/m	data	data if w = 1
Immediate to accumulator	0001010w	data	data if w = 1	

**INC - Increment**

Register/memory	1111111w	mod 000 r/m		
Register	01000	reg		
AAA - ASCII adjust for add	0110111			
DAA - Decimal adjust for add	00100111			

**SUB - Subtract**

Reg. memory and register to either	0010100w	mod reg r/m		
Immediate from register/memory	1000001w	mod 010 r/m	data	data if w = 1
Immediate from accumulator	0010110w	data	data if w = 1	

**SBB - Subtract with borrow**

Reg. memory and register to either	0011100w	mod reg r/m		
Immediate from register/memory	1000001w	mod 011 r/m	data	data if w = 1
Immediate from accumulator	0011110w	data	data if w = 1	

**DIC - Decrement**

Register/memory	1111111w	mod 001 r/m		
Register	01001	reg		
NEG - Change sign	1111011w	mod 011 r/m		

**CMF - Compare**

Register/memory and register	0011100w	mod reg r/m		
Immediate with register/memory	1000001w	mod 111 r/m	data	data if w = 01
Immediate with accumulator	0011110w	data	data if w = 1	
AAS - ASCII adjust for subtract	00111111			
DAS - Decimal adjust for subtract	00101111			
MUL - Multiply (unsigned)	1111011w	mod 100 r/m		
IMUL - Integer multiply (signed)	1111011w	mod 101 r/m		
AAM - ASCII adjust for multiply	11010100	00001010		
DIV - Divide (unsigned)	1111011w	mod 110 r/m		
IDIV - Integer divide (signed)	1111011w	mod 111 r/m		
AAD - ASCII adjust for divide	11010101	00001010		
CWD - Convert byte to word	10011000			
CWB - Convert word to double word	10011001			

**LOGIC**

NOT - Invert	1111011w	mod 010 r/m		
SHL/SAL - Shift logical/arithmetic left	110100w	mod 100 r/m		
SHR - Shift logical right	110100w	mod 101 r/m		
SAR - Shift arithmetic right	110100w	mod 111 r/m		
ROL - Rotate left	110100w	mod 000 r/m		
ROR - Rotate right	110100w	mod 001 r/m		
RCL - Rotate through carry flag left	110100w	mod 010 r/m		
RCR - Rotate through carry right	110100w	mod 011 r/m		

**AND - And**

Reg. memory and register to either	0010000w	mod reg r/m		
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w = 1
Immediate to accumulator	0010010w	data	data if w = 1	

**TEST - And function to flags, no result**

Register memory and register	1000010w	mod reg r/m		
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w = 1
Immediate data and accumulator	1010100w	data	data if w = 1	

**OR - Or**

Reg. memory and register to either	0000100w	mod reg r/m		
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w = 1
Immediate to accumulator	0000110w	data	data if w = 1	

**XOR - Exclusive or**

Reg. memory and register to either	0011000w	mod reg r/m		
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w = 1
Immediate to accumulator	0011010w	data	data if w = 1	

**STRING MANIPULATION**

REP - Repeat	11110011			
MOVS - Move byte/word	1010010w			
CMPS - Compare byte/word	1010011w			
SCAS - Scan byte/word	1010111w			
LODS - Load byte/word to AL/AX	1010110w			
STOS - Store byte/word from AL/AX	1010101w			

Instruction Set Summary (continued)

**CONTROL TRANSFER**

**CALL - Call**

	7 8 5 4 3 2 1 0	7 8 5 4 3 2 1 0	7 8 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg low	seg high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	

**JMP - Unconditional Jump**

Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within segment short	1 1 1 0 1 0 1 1	disp	
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg low	seg high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	

**RET - Return from CALL**

Within segment	1 1 0 0 0 0 1 1		
Within seg. adding immed to SP	1 1 0 0 0 0 1 0	data low	data high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0	data low	data high

**JE/JZ - Jump on equal/zero**

JL/JBE - Jump on less/not greater or equal	0 1 1 1 0 1 0 0	disp	
JLE/JBE - Jump on less or equal/not greater	0 1 1 1 1 1 0 0	disp	
JLE/JBE - Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp	
JB/JBAE - Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp	
JBE/JBA - Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp	
JP/JPE - Jump on parity/parity even	0 1 1 1 1 0 1 0	disp	
JO - Jump on overflow	0 1 1 1 0 0 0 0	disp	
JS - Jump on sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ - Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE - Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp	
JNLE/JE - Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp	

**JNB/JAE - Jump on not below/above or equal**

	7 8 5 4 3 2 1 0	7 8 5 4 3 2 1 0
JNB/JAE	0 1 1 1 0 0 1 1	disp
JBE/JA - Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JBP/JPB - Jump on not parity/par odd	0 1 1 1 1 0 1 1	disp
JNO - Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS - Jump on not sign	0 1 1 1 1 0 0 1	disp
LOOP - Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE - Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE - Loop while not zero/equal	1 1 1 0 0 0 0 0	disp
JCZ - Jump on CX zero	1 1 1 0 0 0 1 1	disp

**INT - Interrupt**

Type specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INT0 - Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET - Interrupt return	1 1 0 0 1 1 1 1	

**PROCESSOR CONTROL**

CLC - Clear carry	1 1 1 1 1 0 0 0
CMC - Complement carry	1 1 1 1 0 1 0 1
STC - Set carry	1 1 1 1 1 0 0 1
CLD - Clear direction	1 1 1 1 1 1 0 0
STD - Set direction	1 1 1 1 1 1 0 1
CLI - Clear interrupt	1 1 1 1 1 0 1 0
STI - Set interrupt	1 1 1 1 1 0 1 1
HLT - Halt	1 1 1 1 0 1 0 0
WAIT - Wait	1 0 0 1 1 0 1 1
ESC - Escape i/o external device	1 1 0 1 1 x x x mod x x r/m
LOCK - Bus lock prefix	1 1 1 1 0 0 0 0

**Footnotes:**

- AL - 8-bit accumulator
- AX - 16-bit accumulator
- CX - Count register
- DS - Data segment
- ES - Extra segment
- Above/below refers to unsigned value
- Greater - more positive
- Less - less positive (more negative) signed values
- if d = 1 then "to" reg; if d = 0 then "from" reg
- if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field  
 if mod = 00 then DISP = 0\*, disp-low and disp-high are absent  
 if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent  
 if mod = 10 then DISP = disp-high disp-low  
 if r/m = 000 then EA = (BX) + (SI) + DISP  
 if r/m = 001 then EA = (BX) + (DI) + DISP  
 if r/m = 010 then EA = (BP) + (SI) + DISP  
 if r/m = 011 then EA = (BP) + (DI) + DISP  
 if r/m = 100 then EA = (SI) + DISP  
 if r/m = 101 then EA = (DI) + DISP  
 if r/m = 110 then EA = (BP) + DISP\*  
 if r/m = 111 then EA = (BX) + DISP  
 DISP follows 2nd byte of instruction (before data if required)

\*except if mod = 00 and r/m = 110 then EA = disp-high disp-low

if s = 01 then 16 bits of immediate data form the operand  
 if s = 11 then an immediate data byte is sign extended to form the 16-bit operand  
 if v = 0 then "count" = 1, if v = 1 then "count" in (CL)  
 x = don't care  
 z is used for string primitives for comparison with ZF FLAG

**SEGMENT OVERRIDE PREFIX**

0 0 1 reg 1 1 0

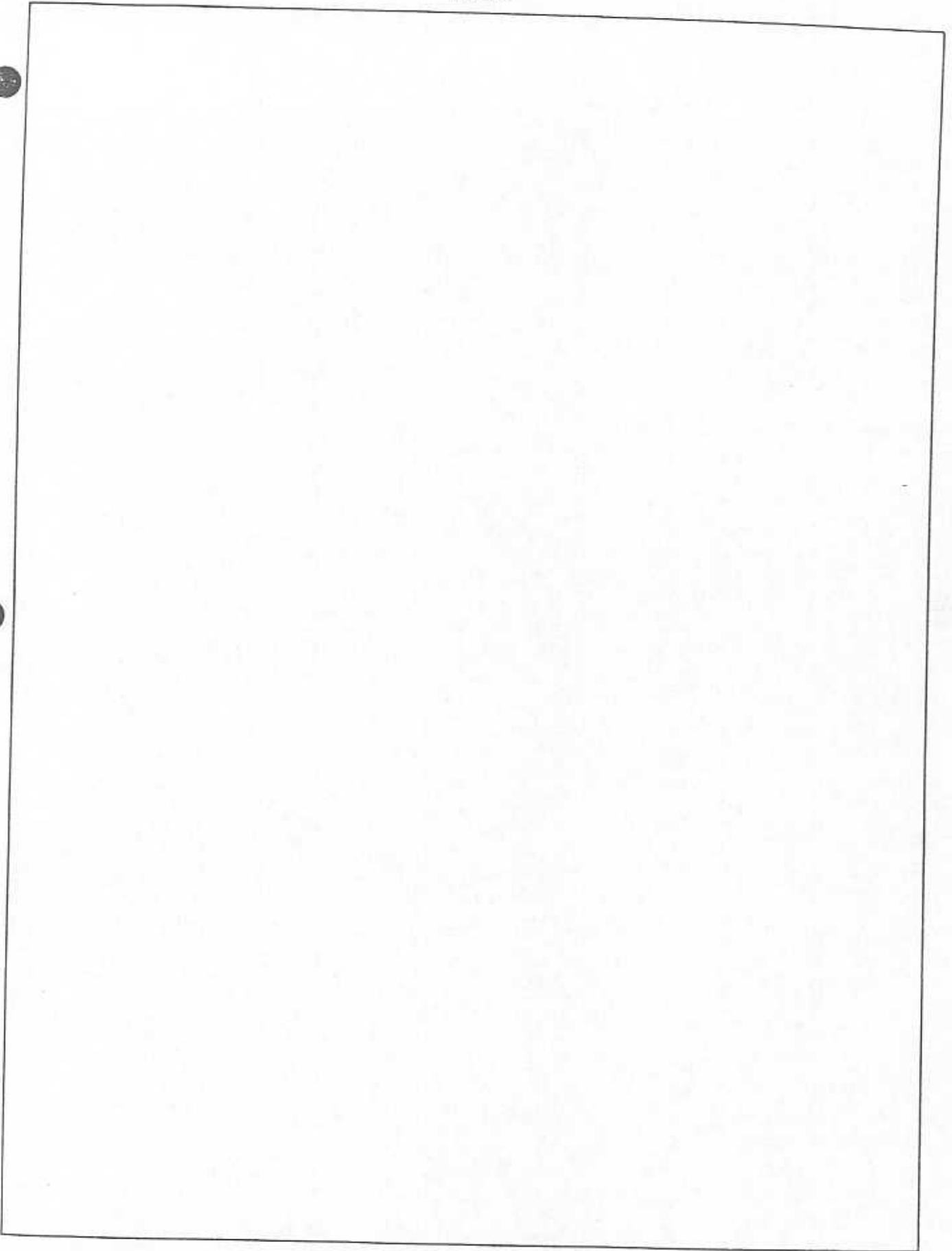
REG is assigned according to the following table

16-Bit [w = 1]	8-Bit [w = 0]	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

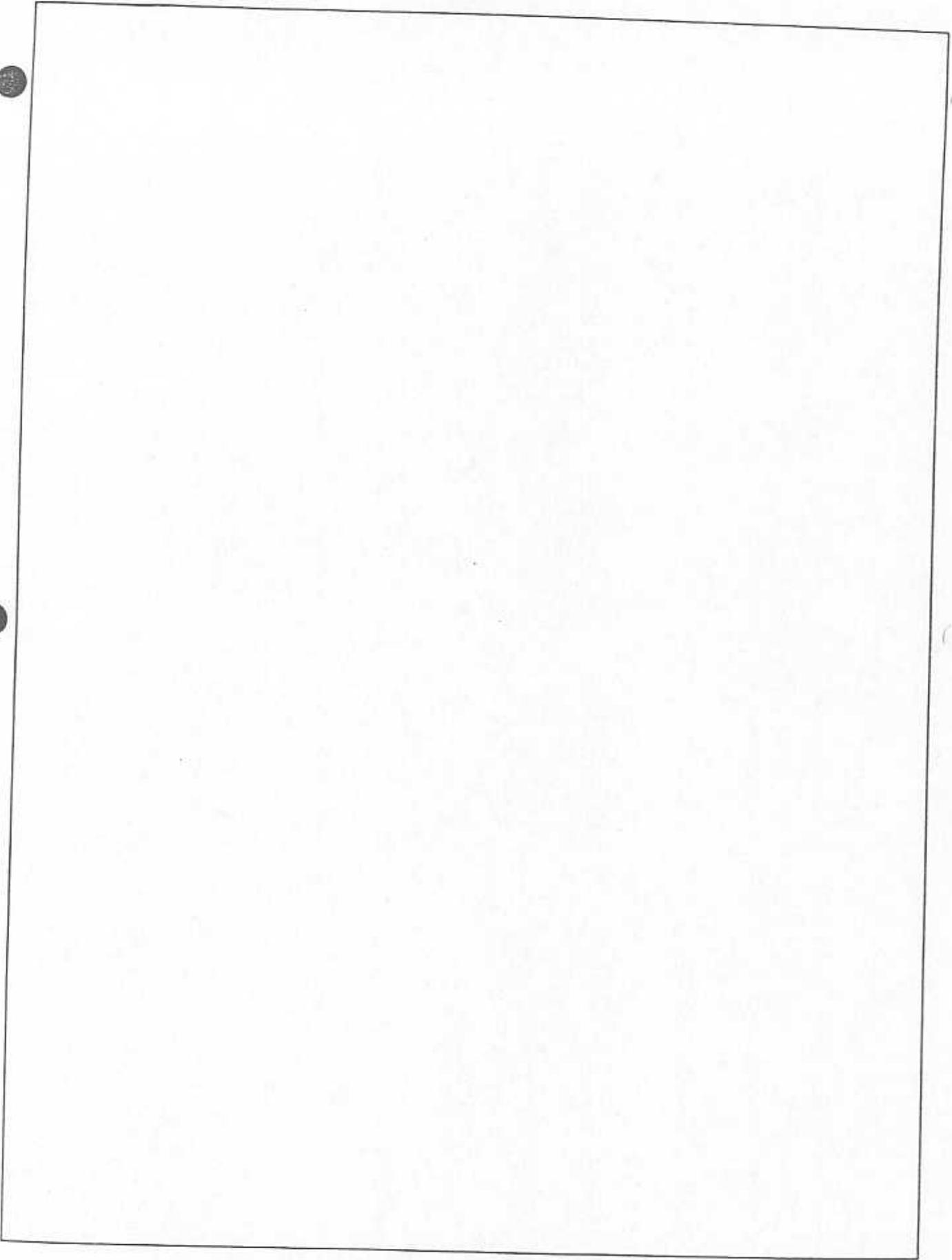
Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file

FLAGS = X X X X (OF) (DF) (IF) (TF) (SF) (ZF) X (AF) X (PF) X (CF)

*Notes*

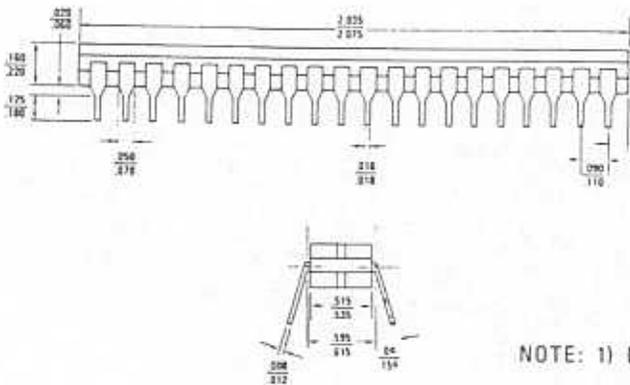


*Notes*

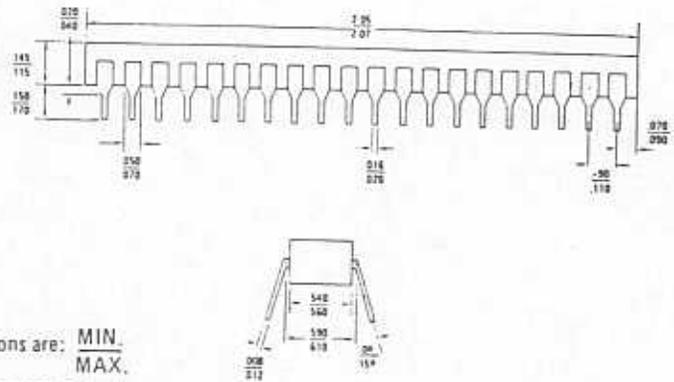


Packaging

40 LEAD CERDIP

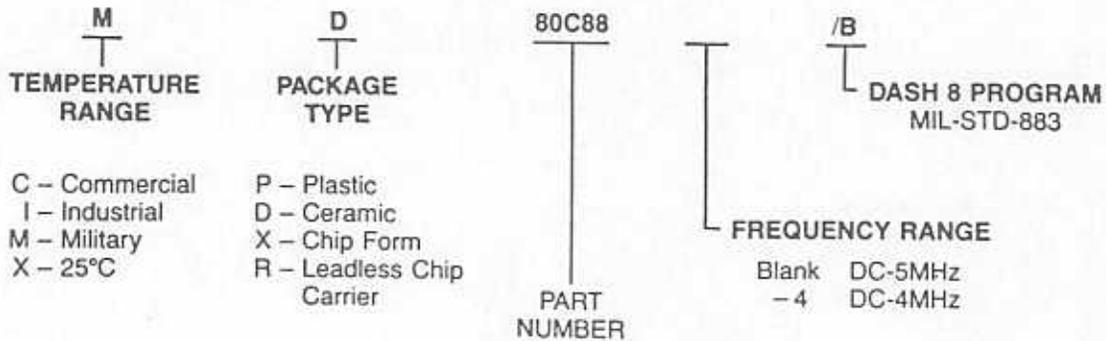


40 LEAD PLASTIC



NOTE: 1) Dimensions are:  $\frac{\text{MIN.}}{\text{MAX.}}$   
 2) All Dimensions in inches

Ordering Information



Sales Offices

P.O. BOX 7065  
 HUNTSVILLE, AL. 35807  
 (205) 837-8886

SUITE 250  
 1717 E. MORTEN AVE.  
 PHOENIX, AZ. 85020  
 (602) 870-0080

SUITE 320  
 1503 SO. COAST DRIVE  
 COSTA MESA, CA. 92626  
 (714) 540-2176

SUITE C100  
 883 STIERLIN ROAD  
 MOUNTAIN VIEW, CA. 94043  
 (415) 964-6443

SUITE 205  
 6400 CANOGA AVE.  
 WOODLAND HILLS, CA. 91367  
 (818) 992-0686

SUITE 215  
 3890 W. COMMERCIAL BLVD.  
 FT. LAUDERDALE, FL. 33309  
 (305) 739-0016

SUITE 400  
 875 JOHNSON FERRY RD.  
 ATLANTA, GA. 30342  
 (404) 256-4000

SUITE 300  
 6400 SHAFER COURT  
 ROSEMONT, IL. 60018  
 (312) 692-4960

SUITE 101  
 1717 EAST 116TH STREET  
 CARMEL, IN. 46032  
 (317) 844-8011

SUITE 308  
 1 BURLINGTON WOODS DR.  
 BURLINGTON, MA. 01803  
 (617) 273-5942

SUITE 703  
 2850 METRO DRIVE  
 MINNEAPOLIS, MN. 55420  
 (612) 854-3558

P.O. BOX 31747  
 RALEIGH, NC. 27622  
 (919) 847-8985

SUITE 426  
 555 BROAD HOLLOW ROAD  
 MELVILLE, NY. 11747  
 (516) 249-4500

SUITE 280  
 300 E. WILSON BRIDGE RD.  
 WORTHINGTON, OH. 43085  
 (614) 885-5967

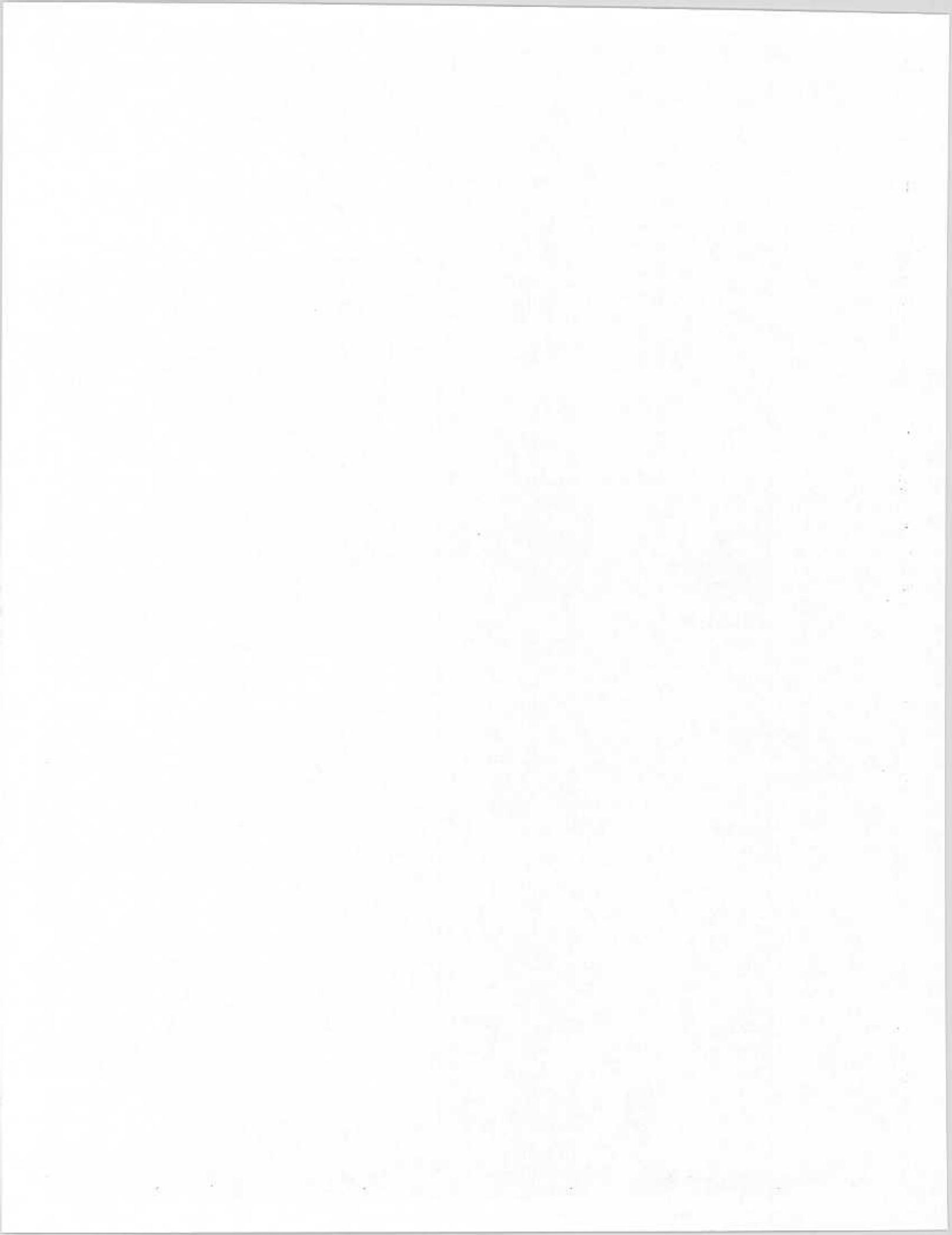
SUITE 1101  
 996 OLD EAGLE SCHOOL RD.  
 WAYNE, PA. 19087  
 (215) 687-6680

SUITE 110  
 17120 DALLAS PARKWAY  
 DALLAS, TX. 75248  
 (214) 248-3237

33919 9TH AVE., SOUTH  
 FEDERAL WAY, WA. 98003  
 (206) 838-4878

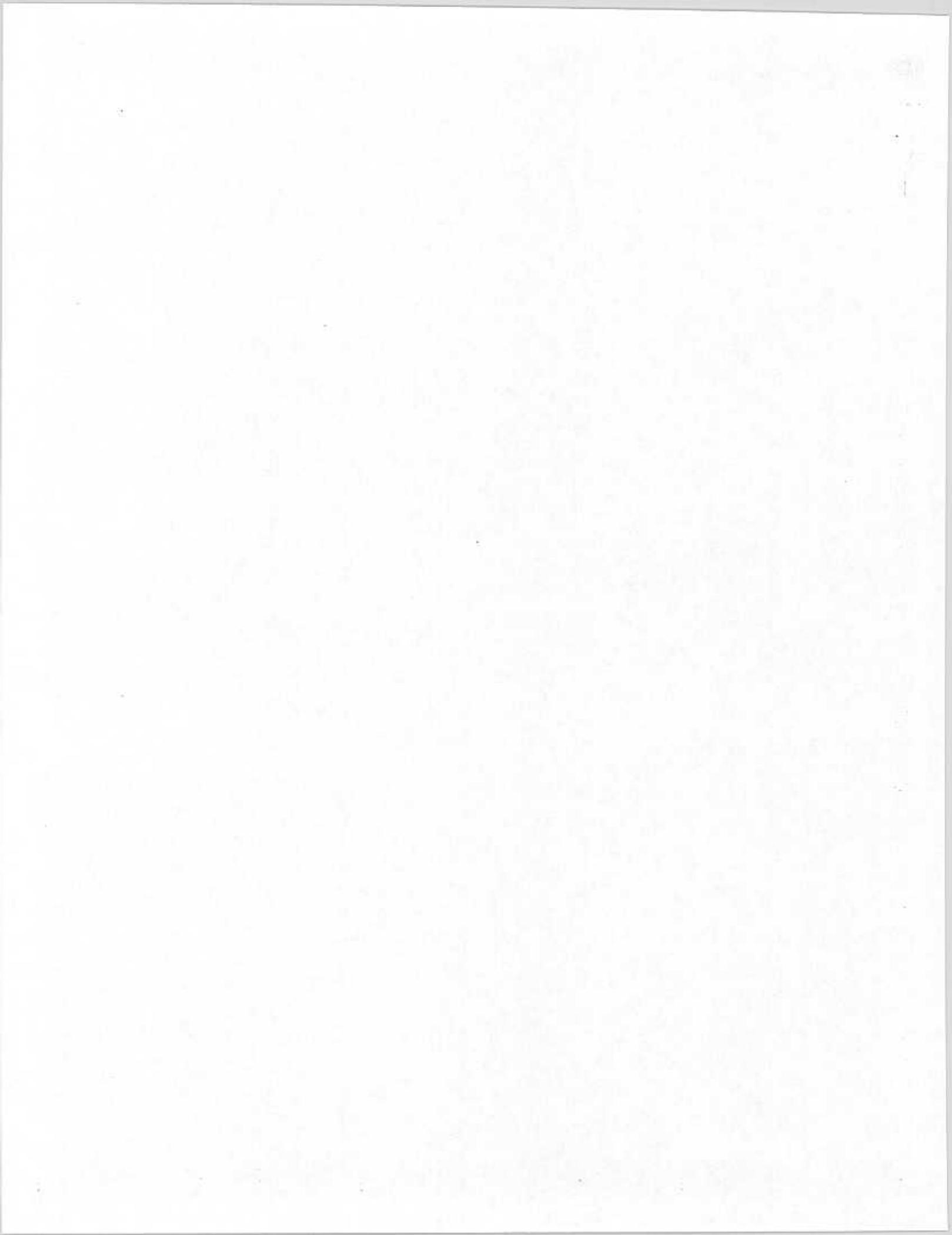
SUITE 8  
 2005 BROADWAY  
 VANCOUVER, WA. 98663  
 (206) 696-0043





8274

MULTI-PROTOCOL SERIAL  
CONTROLLER





## 8274 MULTI-PROTOCOL SERIAL CONTROLLER (MPSC)

- Asynchronous, Byte Synchronous and Bit Synchronous Operation
- Two Independent Full Duplex Transmitters and Receivers
- Fully Compatible with 8048, 8051, 8085, 8088, 8086, 80188 and 80186 CPU's; 8257 and 8237 DMA Controllers; and 8089 I/O Proc.
- 4 Independent DMA Channels
- Baud Rate: DC to 880K Baud
- Asynchronous:
  - 5-8 Bit Character; Odd, Even, or No Parity; 1, 1.5 or 2 Stop Bits
  - Error Detection: Framing, Overrun, and Parity
- Byte Synchronous:
  - Character Synchronization, Int. or Ext.
  - One or Two Sync Characters
  - Automatic CRC Generation and Checking (CRC-16)
  - IBM Bisync Compatible
- Bit Synchronous:
  - SDLC/HDLC Flag Generation and Recognition
  - 8 Bit Address Recognition
  - Automatic Zero Bit Insertion and Deletion
  - Automatic CRC Generation and Checking (CCITT-16)
  - CCITT X.25 Compatible
- Available in EXPRESS
  - Standard Temperature Range

The Intel® 8274 Multi-Protocol Serial Controller (MPSC) is designed to interface High Speed Communications Lines using Asynchronous, IBM Bisync, and SDLC/HDLC protocol to Intel microcomputer systems. It can be interfaced with Intel's MCS-48, -85, -51; iAPX-86, -88, -186 and -188 families, the 8237 DMA Controller, or the 8089 I/O Processor in polled, interrupt driven, or DMA driven modes of operation.

The MPSC is a 40 pin device fabricated using Intel's High Performance HMOS Technology.

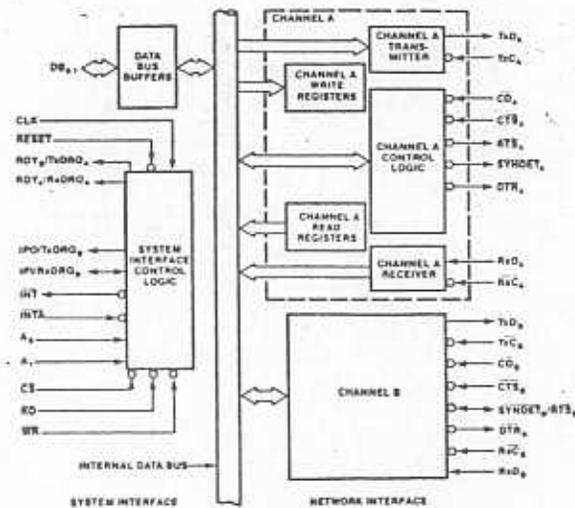


Figure 1. Block Diagram

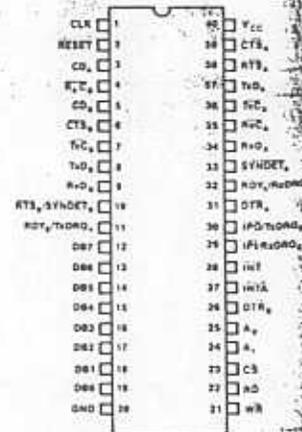


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
CLK	1	I	Clock: System clock, TTL compatible.
RESET	2	I	Reset: A low signal on this pin will force the MPSC to an idle state. Tx <sub>D</sub> and Tx <sub>C</sub> are forced high. The modem interface output signals are forced high. The MPSC will remain idle until the control registers are initialized. Reset must be true for one complete CLK cycle.
CD <sub>A</sub>	3	I	Carrier Detect (Channel A): This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the Rx <sub>D</sub> line. If the auto enable control is set the 8274 will not enable the serial receiver until CD <sub>A</sub> has been activated.
RxC <sub>B</sub>	4	I	Receive Clock (Channel B): The serial data are shifted into the Receive Data input (Rx <sub>D</sub> ) on the rising edge of the Receive Clock.
CD <sub>B</sub>	5	I	Carrier Detect (Channel B): This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the Rx <sub>D</sub> line. If the auto enable control is set the 8274 will not enable the serial receiver until CD <sub>B</sub> has been activated.
CTS <sub>B</sub>	6	I	Clear to Send (Channel B): This interface signal is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until CTS has been activated.
TxC <sub>B</sub>	7	I	Transmit Clock (Channel B): The serial data are shifted out from the Transmit Data output (Tx <sub>D</sub> ) on the falling edge of the Transmit Clock.
TxD <sub>B</sub>	8	O	Transmit Data (Channel B): This pin transmits serial data to the communications channel (Channel B).
RxD <sub>B</sub>	9	I	Receive Data (Channel B): This pin receives serial data from the communications channel (Channel B).
SYNDET <sub>B</sub> / RTS <sub>B</sub>	10	I/O	Synchronous Detection (Channel B): This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel B).  Request to Send (Channel B): General purpose output, generally used to signal that Channel B is ready to send data.  SYNDET <sub>B</sub> or RTS <sub>B</sub> selection is done by WR2, D7, (Channel A)

Symbol	Pin No.	Type	Name and Function
RDY <sub>B</sub> / TxDRQ <sub>A</sub>	11	O	Ready (Channel B)/Transmitter DMA Request (Channel A): In mode 0 this pin is RDY <sub>B</sub> and is used to synchronize data transfers between the processor and the MPSC (Channel B). In modes 1 and 2 this pin is TxDRQ <sub>A</sub> and is used by the Channel A transmitter to request a DMA transfer.
DB7	12	I/O	Data Bus: The Data Bus lines are bidirectional three state lines which interface with the system's Data Bus.
DB6	13		
DB5	14		
DB4	15		
DB3	16		
DB2	17		
DB1	18		
DB0	19		
GND	20		Ground.
V <sub>CC</sub>	40		Power: +5V Supply
CTS <sub>A</sub>	39	I	Clear to Send (Channel A): This interface signal is supplied by the Modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until CTS has been activated.
RTS <sub>A</sub>	38	O	Request to Send (Channel A): general purpose output commonly used to signal that Channel A is ready to send data.
TxD <sub>A</sub>	37	O	Transmit Data (Channel A): This pin transmits serial data to the communications channel (Channel A).
TxC <sub>A</sub>	36	I	Transmit Clock (Channel A): The serial data are shifted out from the Transmit Data output (Tx <sub>D</sub> ) on the falling edge of the Transmit Clock.
RxC <sub>A</sub>	35	I	Receive Clock (Channel A): The serial data are shifted into the Receive Data input (Rx <sub>D</sub> ) on the rising edge of the Receive Clock.
RxD <sub>A</sub>	34	I	Receive Data (Channel A): This pin receives serial data from the communications channel (Channel A).
SYNDET <sub>A</sub>	33	I/O	Synchronous Detection (Channel A): This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel A).
RDY <sub>A</sub> / RxDRQ <sub>B</sub>	32	O	Ready: In mode 0 this pin is RDY <sub>A</sub> and is used to synchronize data transfers between the processor and the MPSC (Channel A). In modes 1 and 2 this pin is RxDRQ <sub>B</sub> and is used by the channel A receiver to request a DMA transfer.
DTR <sub>A</sub>	31	O	Data Terminal Ready (Channel A): General purpose output

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
IPO/ TxDRQ <sub>B</sub>	30	O	Interrupt Priority Out/Transmitter DMA Request (Channel B): In modes 0 and 1, this pin is Interrupt Priority Out. It is used to establish a hardware interrupt priority scheme with $\overline{IP}$ . It is low only if $\overline{IP}$ is low and the controlling processor is not servicing an interrupt from this MPSC. In mode 2 it is TxDRQ <sub>B</sub> and is used to request a DMA cycle for a transmit operation (Channel B).
$\overline{IP}$ / RxDRQ <sub>B</sub>	29	I/O	Interrupt Priority In/Receiver DMA Request (Channel B): In modes 0 and 1, $\overline{IP}$ is Interrupt Priority In. A low on $\overline{IP}$ means that no higher priority device is being serviced by the controlling processor's interrupt service routine. In mode 2 this pin is RxDRQ <sub>B</sub> and is used to request a DMA cycle for a receive operation (Channel B). In interrupt mode, this pin must be tied low.
INT	28	O	Interrupt: The interrupt signal indicates that the highest priority internal interrupt requires service (open collector). Priority can be resolved via an external interrupt controller or a daisy-chain scheme.

Symbol	Pin No.	Type	Name and Function
INTA	27	I	Interrupt Acknowledge: This Interrupt Acknowledge signal allows the highest priority interrupting device to generate an interrupt vector. This pin must be pulled high (inactive) in non-vector mode.
DTR <sub>B</sub>	26	O	Data Terminal Ready (Channel B): This is a general purpose output.
A <sub>0</sub>	25	I	Address: This line selects Channel A or B during data or command transfers. A low selects Channel A.
A <sub>1</sub>	24	I	Address: This line selects between data or command information transfer. A low means data.
CS	23	I	Chip Select: This signal selects the MPSC and enables reading from or writing into its registers.
RD	22	I	Read: Read controls a data byte or status byte transfer from the MPSC to the CPU.
WR	21	I	Write: Write controls transfer of data or commands to the MPSC.

## RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ( $\overline{RTS}$ ,  $\overline{DTR}$ ) are forced high. In addition, the pointers registers are set to zero.

## GENERAL DESCRIPTION

The Intel 8274 Multi-Protocol Serial Controller is a microcomputer peripheral device which supports Asynchronous, Byte Synchronous (Monosync, IBM Bisync), and Bit Synchronous (ISO's HDLC, IBM's SDLC) protocols. This controller's flexible architecture allows easy implementation of many variations of these three protocols with low software and hardware overhead.

The Multi-Protocol Serial controller (MPSC) implements two independent serial receiver/transmitter channels.

The MPSC supports several microprocessor interface options: Polled, Wait, Interrupt driven and DMA driven. The MPSC is designed to support INTEL'S MCS-85 and IAPX 86, 88, 186, 188 families.

## FUNCTIONAL DESCRIPTION

Additional information on Asynchronous and Synchronous Communications with the 8274 is available respectively in the Applications Notes AP 134 and AP 145.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B).

In the following discussion, the writable registers will be referred to as WRO through WR7 and the readable registers will be referred to as RRO through RR2.

This section of the data sheet describes how the Asynchronous and Synchronous protocols are implemented in the MPSC. It describes general considerations, transmit operation, and receive operation for Asynchronous, Byte Synchronous, and Bit Synchronous protocols.

## ASYNCHRONOUS OPERATIONS

### TRANSMITTER/RECEIVER INITIALIZATION

(See Detailed Command Description Section for complete information)

In order to operate in asynchronous mode, each MPSC channel must be initialized with the following information:

1. **Transmit/Receive Clock Rate.** This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 1, 16, 32, or 64 times the data-link bit rate. If the X1 clock mode is selected, the bit synchronization must be accomplished externally.
2. **Number of Stop Bits.** This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1 1/2, or 2.
3. **Parity Selection.** Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4.
4. **Receiver Character Length.** This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3.
5. **Receiver Enable.** The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3.
6. **Transmitter Character Length.** This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 0).

The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

D7	D6	D5	D4	D3	D2	D1	D0	Number of Bits Transmitted (Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. **Transmitter Enable.** The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5.

### 8. Interrupt Mode.

For data transmission via a modem or RS-232-C interface, the following information must also be specified:

1. The Request To Send (RTS) (WR5; D1) and Data Terminal Ready (DTR) (WR5; D7) bits must be set along with the Transmit Enable bit (WR5; D3).
2. Auto Enable may be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. However, the Transmit Enable bit (WR3; D3) and Receive Enable bit (WR3; D1) must be set in order to use the Auto Enable mode. Auto Enable is controlled by bit 5 of WR3.

When loading Initialization parameters into the MPSC, WR4 information must be written before the WR1, WR3, WR5 parameters commands.

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSC\$RX\$INIT) for asynchronous communication is listed in Intel Application Note AP 134.

## TRANSMIT

The transmit function begins when the Transmit Enable bit (WR5; D3) is set. The MPSC automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits (1, 1.5 or 2 bits) to the data character being transmitted. 1.5 stop bits option must be used with X16, X32 or X64 clock options only.

The serial data are shifted out from the Transmit Data (TxD) output on the falling edge of the Transmit Clock (TxC) input at a rate programmable to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the TxC input.

The TxD output is held high when the transmitter has no data to send, unless, under program control, the Send Break (WR5; D4) command is issued to hold the TxD low.

If the External/Status Interrupt bit (WR1; D0) is set, the status of CD, CTS and SYNDET are monitored and, if any changes occur for a period of time greater than the minimum specified pulse width, an interrupt is generated. CTS is usually monitored using this interrupt feature (e.g. Auto Enable option).

The Transmit Buffer Empty bit (RRO; D2) is set by the MPSC when the data byte from the buffer is loaded in the transmit shift register. Data should be written to the MPSC only when the Tx buffer becomes empty to prevent overwriting.

### Receive

The receive function begins when the Receive Enable (WR3; D0) bit is set. If the Auto Enable (WR3; D5) option is selected, then Carrier Detect (CD) must also be low. A valid start bit is detected if a low persists for at least 1/2 bit time on the Receive Data (RxD) input.

The data is sampled at mid-bit time, on the rising edge of RxC, until the entire character is assembled. The receiver inserts 1's when a character is less than 8 bits. If parity (WR4; D0) is enabled and the character is less than 8 bits the parity bit is not stripped from the character.

### Error Reporting

The receiver also stores error status for each of the 3 data characters in the data buffer. Three error conditions may be encountered during data reception in the asynchronous mode:

1. **Parity.** If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4). When a parity error is detected, the parity error flag (RR1; D4) is set and remains set until it is reset by the Error Reset command (WR0; D5, D4, D3).

2. **Framing.** A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled). When a Framing Error is detected, the Framing Error bit (RR1; D6) is set. The detection of a Framing Error adds an additional 1/2 bit time to the character time so the Framing Error is not interpreted as a new start bit.

3. **Overrun.** If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1; D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. Only the overwritten character is flagged with the Receive Overrun bit. The Receive Overrun bit (RR1; D5) is reset by the Error Reset command (WR0; D5, D4, D3).

### External/Status Latches

The MPSC continuously monitors the state of five external/status conditions:

1. CTS — clear-to-send input pin.
2. CD — carrier-detect input pin.
3. SYNDET — sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK — a break condition (series of space bits on the receiver input pin).
5. Tx UNDERRUN/EOM — Transmitter Underrun/End of Message.

A change of state in any of these monitored conditions will cause the associated status bit in RR0 to be latched (and optionally cause an interrupt).

If the External/Status Interrupt bit (WR1; D0) is enabled, Break Detect (RR0; D7) and Carrier Detect (RR0; D3) will cause an interrupt. Reset External/Status interrupts (WR0; D5, D4, D3) will clear Break Detect and Carrier Detect bits if they are set.

Asynchronous Mode Register Setup

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	0	0	0	0	Rx ENABLE
WR4	00 X1 Clock 01 X16 Clock 10 X32 Clock 11 X64 Clock		0	0	01 1 STOP BIT 10 1½ STOP BITS 11 2 STOP BITS		EVEN/ ODD PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0	RTS	0

**SYNCHRONOUS OPERATION—  
MONOSYNC, BISYNC**

**General**

The MPSC must be initialized with the following parameters: odd or even parity (WR4; D1, D0), X1 clock mode (WR4; D7, D6), 8- or 16-bit sync character (WR4; D5, D4), CRC polynomial (WR5; D2), Transmitter Enable (WR5; D3), interrupt modes (WR1, WR2), transmit character length (WR5; D6, D5) and receive character length (WR3; D7, D6). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The data is transmitted on the falling edge of the Transmit Clock, (TxC) and is received on the rising edge of Receive Clock (RxC). The X1 clock is used for both transmit and receive operations for all three sync modes: Mono, Bi and External.

**Transmit Set-Up—Monosync, Bisync**

Transmit data is held high after channel reset, or if the transmitter is not enabled. A break may be programmed to generate a spacing line that begins as soon as the Send Break (WR5; D4) bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

Using interrupts for data transfer requires that the Transmit Interrupt/DMA Enable bit (WR1; D1) be set. An interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied

Synchronous Mode Register Setup—Monosync, Bisync

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	Rx ENABLE
WR4	0	0	00 8 bit Sync 01 16 bit Sync 11 Ext Sync		0	0	EVEN/ ODD PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	1 (SELECTS CRC-16)	RTS	Tx CRC ENABLE

either by writing another character into the transmitter or by resetting the Transmitter Interrupt/DMA Pending latch with a Reset Transmitter Interrupt/DMA Pending Command (WR0; D5, D4, D3). If nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupt, but this situation does cause a Transmit Underrun condition (RR0; D6).

Data Transfers using the RDY signal are for software controlled data transfers such as block moves. RDY tells the CPU that the MPSC is not ready to accept/provide data and that the CPU must extend the output/input cycle. DMA data transfers use the TxDRQ A/B signals which indicate that the transmit buffer is empty, and that the MPSC is ready to accept the next data character. If the data character is not loaded into the MPSC by the time the transmit shift register is empty, the MPSC enters the Transmit Underrun condition.

The MPSC has two programmable options for solving the transmit underrun condition: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0; D6) is in a set condition allowing the insertion of sync characters when there is no data to send. The CRC is not calculated on these automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data to send.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded in the Transmit Shift Register. The status register indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded into the Tx Shift Register). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5; D3).

**Bisync CRC Generation.** Setting the Transmit CRC enable bit (WR5; D0) indicates CRC accumulation when the program sends the first data character to

the MPSC. Although the MPSC automatically transmits up to two sync characters (16 bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The Transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded into the transmit shift register. To ensure this bit in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the MPSC.

**Transmit Transparent Mode.** Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16 bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the MPSC.

In the transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16) (WR4; D5, D4). When in the Monosync mode, the transmitter sends from WR6 and the receiver compares against WR7. One of two CRC polynomials, CRC 16 or SDLC, may be used with synchronous modes. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command (WR0; D7, D6).

The External/Status interrupt (WR1; D0) mode can be used to monitor the status of the CTS input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enable (WR3; D5) feature can be used to enable the transmitter when CTS is active. The first data transfer to the MPSC can begin when the External/Status interrupt occurs (CTS (RR0; D5) status bit set) following the Transmit Enable command (WR5; D3).

## Receive

After a channel reset, the receiver is in the Hunt phase, during which the MPSC looks for character synchronization. The Hunt begins only when the receiver is enabled and data transfer begins only when character synchronization has been achieved. If character synchronization is lost, the hunt phase can be re-entered by writing the Enter Hunt Phase (WR3; D4) bit. The assembly of received data continues until the MPSC is reset or until the receiver is

disabled (by command or by  $\overline{CD}$  while in the Auto Enables mode) or until the CPU sets the Enter Hunt Phase bit. Under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit (WR3; D1) bit. After character synchronization is achieved the assembled characters are transferred to the receive data FIFO. After receiving the first data character, the Sync Character Load Inhibit bit should be reset to zero so that all characters are received, including the sync characters. This is important because the received CRC may look like a sync character and not get received.

Data may be transferred with or without interrupts. Transferring data without interrupts is used for a purely polled operation or for off-line conditions. There are three interrupt modes available for data transfer: Interrupt on First Character Only, Interrupt on Every Character, and Special Receive Conditions Interrupt.

Interrupt on First Character Only mode is normally used to start a polling loop, a block transfer sequence using RDY to synchronize the CPU to the incoming data rate, or a DMA transfer using the RxDQ signal. The MPSC interrupts on the first character and thereafter only interrupts after a Special Receive Condition is detected. This mode can be reinitialized using the Enable Interrupt On Next Receive Character (WR0; D5, D4, D3) command which allows the next character received to generate an interrupt. Parity Errors do not cause interrupts, but End of Frame (SDLC operation) and Receive Overrun do cause interrupts in this mode. If the external status interrupts (WR1; D0) are enabled an interrupt may be generated any time the  $\overline{CD}$  changes state.

Interrupt On Every Character mode generates an interrupt whenever a character enters the receive

buffer. Errors and Special Receive Conditions generate a special vector if the Status Affects Vector (WR1B; D2) is selected. Also the Parity Error may be programmed (WR1; D4, D3) not to generate the special vector while in the Interrupt On Every Character mode.

The Special Receive Condition interrupt can only occur while in the Receive Interrupt On First Character Only or the Interrupt On Every Receive Character modes. The Special Receive Condition interrupt is caused by the Receive Overrun (RR1; D5) error condition. The error status reflects an error in the current word in the receive buffer, in addition to any Parity or Overrun errors since the last Error Reset (WR0; D5, D4, D3). The Receive Overrun and Parity error status bits are latched and can only be reset by the Error Reset (WR0; D5, D4, D3) command.

The CRC check result may be obtained by checking for CRC bit (RR1; D6). This bit gives the valid CRC result 16 bit times after the second CRC byte has been read from the MPSC. After reading the second CRC byte, the user software must read two more characters (may be sync characters) before checking for CRC result in RR1. Also for proper CRC computation by the receiver, the user software must reset the Receive CRC Checker (WR0; D7, D6) after receiving the first valid data character. The receive CRC Enable bit (WR3; D3) may also be enabled at this time.

## SYNCHRONOUS OPERATION—SDLC

### General

Like the other synchronous operations the SDLC mode must be initialized with the following parameters: SDLC mode (WR4; D5, D4), SDLC polynomial (WR5; D2), Request to Send, Data Terminal Ready,

Synchronous Mode Register Setup—SDLC/HDLC

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5b/char 01 Rx 7b/char 10 Rx 6b/char 11 Rx 8b/char		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1	0	0	0	0	0
			(SELECTS SDLC/ HDLC MODE)					
WR5	DTR	00 Tx 5b/char 01 Tx 7b/char 10 Tx 6b/char 11 Tx 8b/char		SEND BREAK	Tx ENABLE	0 (SELECTS SDLC/ HDLC CRC)	RTS	Tx CRC ENABLE



transmit character length (WR5; D6, D5), interrupt modes (WR1; WR2), Transmit Enable (WR5; D3), Receive Enable (WR3; D0), Auto Enable (WR3; D5) and External/Status Interrupt (WR1; D0). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The Interrupt modes for SDLC operation are similar to those discussed previously in the synchronous operations section.

### Transmit

After a channel reset, the MPSC begins sending SDLC flags.

Following the flags in an SDLC operation the 8-bit address field, control field and information field may be sent to the MPSC by the microprocessor. The MPSC transmits the Frame Check Sequence using the Transmit Underrun feature. The MPSC automatically inserts a zero after every sequence of 5 consecutive 1's except when transmitting Flags or Aborts.

SDLC—like protocols do not have provision for fill characters within a message. The MPSC therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by sending the two bytes of CRC and then one or more flags. This allows very high-speed transmissions under DMA or CPU control without requiring the CPU to respond quickly to the end-of-message situation.

After a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Flag characters are sent. The MPSC begins to send the frame when data is written into the transmit buffer. Between the time the first data byte is written, and the end of the message, the Reset Transmit Underrun/EOM (WR0; D7, D6) command must be issued. The Transmit Underrun/EOM status bit (RR0; D6) is in the reset state at the end of the message which automatically sends the CRC characters.

The MPSC may be programmed to issue a send Abort command (WR0; D5, D4, D3). This command causes at least eight 1's but less than fourteen 1's to be sent before the line reverts to continuous flags.

### Receive

After initialization, the MPSC enters the Hunt phase, and remains in the Hunt phase until the first Flag is received. The MPSC never again enters the Hunt phase unless the microprocessor writes the Enter Hunt command. The MPSC will also detect flags separated by a single zero. For example, the bit pattern 011111101111110 will be detected as two flags.

The MPSC can be programmed to receive all frames or it can be programmed to the Address Search Mode. In the Address Search Mode, only frames with addresses that match the value in WR6 or the global address (0FFH) are received by the MPSC. Extended address recognition must be done by the microprocessor software.

The control and information fields are received as data.

SDLC/HDLC CRC calculation does not have an 8-bit delay, since all characters are included in the calculation, unlike Byte Synchronous Protocols.

Reception of an abort sequence (7 or more 1's) will cause the Break/Abort bit (RR0; D7) to be set and will cause an External/Status interrupt, if enabled. After the Reset External/Status Interrupts Command has been issued, a second interrupt will occur at the end of the abort sequence.

### MPSC

#### Detailed Command/Status Description

##### GENERAL

The MPSC supports an extremely flexible set of serial and system interface modes.

The system interface to the CPU consists of 8 ports or buffers:

CS	A <sub>1</sub>	A <sub>2</sub>	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedance	High Impedance

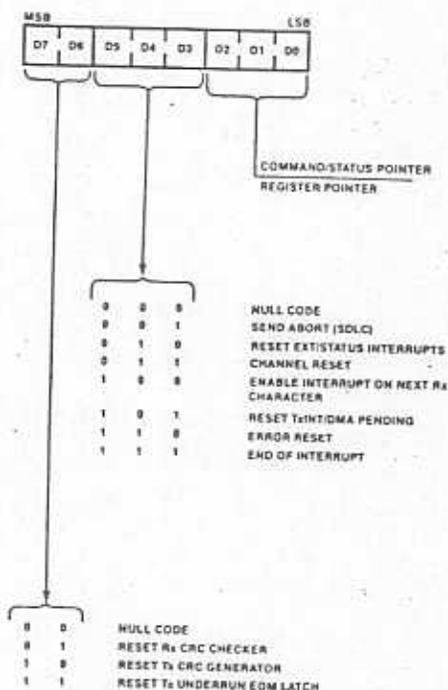
Data buffers are addressed by A<sub>1</sub> = 0, and Command ports are addressed by A<sub>1</sub> = 1.

**COMMAND/STATUS DESCRIPTION**

The following command and status bytes are used during initialization and execution phases of operation. All Command/Status operations on the two channels are identical, and independent, except where noted.

**Detailed Register Description**

Write Register 0 (WR0):



**WR0**

D2, D1, D0—Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5, D4, D3—Command bits determine which of the basic seven commands are to be performed.

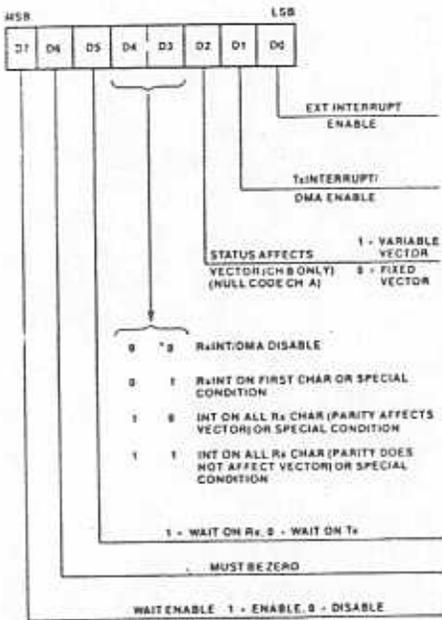
- Command 0 Null—has no effect.
- Command 1, Send Abort—causes the generation of eight to thirteen 1's when in the SDLC mode.
- Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and re-enables them, allowing interrupts to occur again.
- Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.
- Command 4 Enable Interrupt on Next Receive Character—if the Interrupt on First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.
- Command 5 Reset Transmitter Interrupt/DMA Pending—if The Transmit Interrupt/DMA Enable mode is selected, the MPSC automatically interrupts or requests DMA data transfer when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts or DMA requests until the next character has been completely sent.
- Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.
- Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.
- D7, D6 CRC Reset Code
- 00 Null—has no effect.
- 01 Reset Receive CRC Checker—resets the CRC checker to 0's. If in SDLC mode the CRC checker is initialized to all 1's.

- 10 Reset Transmit CRC Generator —resets the CRC generator to 0's. If in SDLC mode the CRC generator's initialized to all 1's.
- 11 Reset Tx Underrun/End of Message Latch.

D1 Transmitter Interrupt/DMA Enable —allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

Write Register 1 (WR1):



D4, D3

Receive Interrupt Mode

0 0

Receive Interrupts/DMA Disabled

0 1

Receive Interrupt on First Character Only or Special Condition

1 0

Interrupt on All Receive Characters or Special Condition (Parity Error is a Special Receive Condition)

1 1

Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5

Wait on Receive/Transmit—when the following conditions are met the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, CS = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY<sub>A</sub> and RDY<sub>B</sub> may be wired OR connected since only one signal is active at any one time while the other is in the High Z state.

D6

Must be Zero

D7

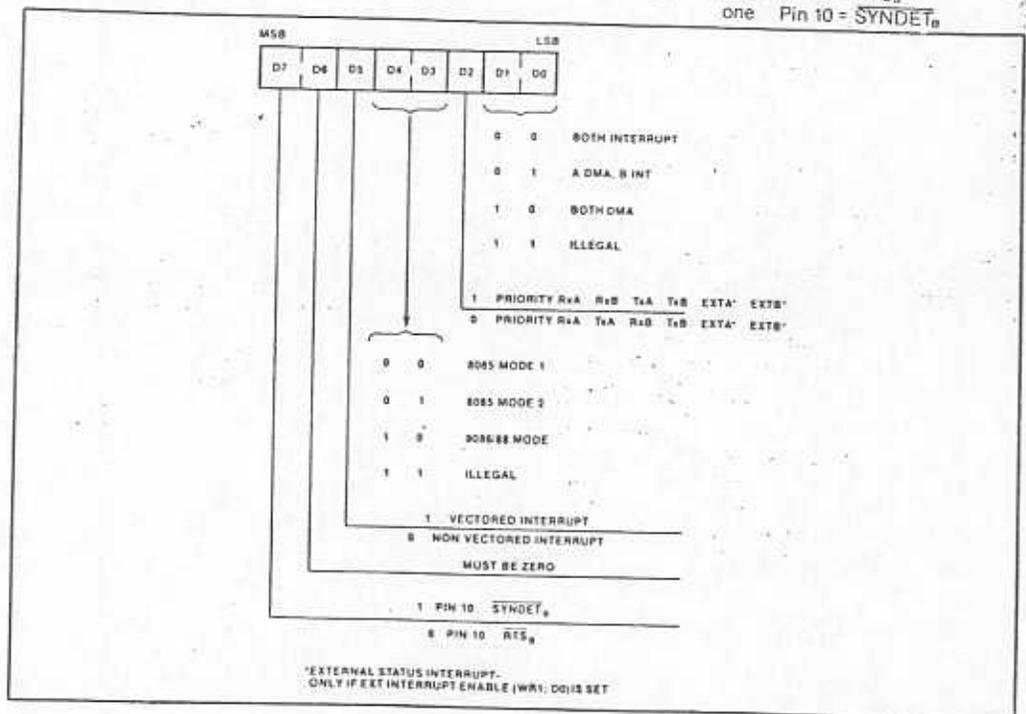
Wait Enable—enables the wait function.

WR1

D0 External/Status Interrupt Enable —allows interrupt to occur as the result of transitions on the CD, CTS or SYNDET inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

<b>WR2</b>	<b>Channel A</b>	<b>D5, D4, D3</b>	<b>Interrupt Code</b> —specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table).
<b>D1, D0</b>	<b>System Configuration</b> —These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.	<b>0 X X</b>	<b>Non-vectored interrupts</b> —intended for use with external DMA CONTROLLER. The Data Bus remains in a high impedance state during INTA sequences.
<b>0 0</b>	Channel A and Channel B both use interrupts	<b>1 0 0</b>	<b>8085 Vector Mode 1</b> —intended for use as the primary MPSC in a daisy chained priority structure. (See System Interface section)
<b>0 1</b>	Channel A uses DMA, Channel B uses interrupt	<b>1 0 1</b>	<b>8085 Vector Mode 2</b> —intended for use as any secondary MPSC in a daisy chained priority structure. (See System Interface section)
<b>1 0</b>	Channel A and Channel B both use DMA	<b>1 1 0</b>	<b>8086/88 Vector Mode</b> —intended for use as either a primary or secondary in a daisy chained priority structure. (See System Interface section)
<b>1 1</b>	Illegal Code	<b>D6</b>	Must be zero.
<b>D2</b>	<b>Priority</b> —this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.	<b>D7</b>	zero Pin 10 = $\overline{RTS}_n$ one Pin 10 = $\overline{SYNDET}_n$
<b>0</b>	(Highest) RxA, TxA, RxB, TxB ExTA, ExTB (Lowest)		
<b>1</b>	(Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest)		

Write Register 2 (WR2): Channel A



The following table describes the MPSC's response to an interrupt acknowledge sequence:

D5	D4	D3	$\overline{IP1}$	MODE	INTA	Data Bus
0	X	X	X	Non-vector	Any INTA	D7 High Impedance D0
1	0	0	0	85 Mode 1	1st INTA 2nd INTA 3rd INTA	1 1 0 0 1 1 0 1 V7 V6 V5 V4* V3* V2* V1 V0 0 0 0 0 0 0 0 0
1	0	0	1	85 Mode 1	1st INTA 2nd INTA 3rd INTA	1 1 0 0 1 1 0 1 High Impedance High Impedance
1	1	0	0	86 Mode	1st INTA 2nd INTA	High Impedance V7 V6 V5 V4 V3 V2* V1*V0*
1	0	1	0	85 Mode 2	1st INTA 2nd INTA 3rd INTA	High Impedance V7 V6 V5 V4* V3* V2* V1 V0 0 0 0 0 0 0 0 0
1	0	1	1	85 Mode 2	1st INTA 2nd INTA 3rd INTA	High Impedance High Impedance High Impedance
1	1	0	1	86 Mode	1st INTA 2nd INTA	High Impedance High Impedance

\*These bits are variable if the "status affects vector" mode has been programmed, (WR1B, D2)

Interrupt/DMA Mode, Pin Functions, and Priority

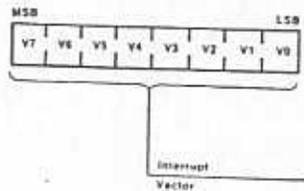
Ch. A WR2			Int/DMA Mode		Pin Functions				Priority	
D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	CH. A	CH. B	RDY <sub>A</sub> / RxDRQ <sub>A</sub> Pin 32	RDY <sub>B</sub> / TxDRQ <sub>A</sub> Pin 11	$\overline{IP1}$ / RxDRQ <sub>B</sub> Pin 29	$\overline{IPO}$ / TxDRQ <sub>B</sub> Pin 30	Highest	Lowest
0	0	0	INT	INT	RDY <sub>A</sub>	RDY <sub>B</sub>	$\overline{IP1}$	$\overline{IPO}$	RxA, TxA, RxB, TxB, EXT <sub>A</sub> , EXT <sub>B</sub>	
1	0	0	INT	INT					RxA, RxB, TxA, TxB, EXT <sub>A</sub> , EXT <sub>B</sub>	
0	0	1	DMA		RxDRQ <sub>A</sub>	TxDRQ <sub>A</sub>	$\overline{IP1}$	$\overline{IPO}$	RxA, TxA (DMA)	
1	0	1	DMA	INT					RxA, RxB, TxB, EXT <sub>A</sub> , EXT <sub>B</sub> (INT)	
0	1	0	DMA		RxDRQ <sub>A</sub>	TxDRQ <sub>A</sub>	RxDRQ <sub>B</sub>	TxDRQ <sub>B</sub>	RxA, TxA (DMA)	
1	1	0	DMA	DMA					RxA, RxB, TxB, EXT <sub>A</sub> , EXT <sub>B</sub> (INT)	
0	1	1	DMA		RxDRQ <sub>A</sub>	TxDRQ <sub>A</sub>	RxDRQ <sub>B</sub>	TxDRQ <sub>B</sub>	RxA, TxA, RxB, TxB (DMA)	
1	1	1	DMA	DMA					RxA, RxB, TxA, TxB, EXT <sub>A</sub> , EXT <sub>B</sub> (INT)	

<sup>1</sup>Special Receive Condition

Interrupt Vector Mode Table

8085 Modes 8086/88 Mode	V <sub>4</sub> V <sub>2</sub>	V <sub>3</sub> V <sub>1</sub>	V <sub>2</sub> V <sub>0</sub>	Channel	Condition
Note 1: Special Receive Condition = Parity Error, Rx Overrun Error, Framing Error, End of Frame (SDLC)	0	0	0	B	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)
	0	0	1		
	0	1	0		
	0	1	1		
	1	0	0	A	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)
	1	0	1		
	1	1	0		
	1	1	1		

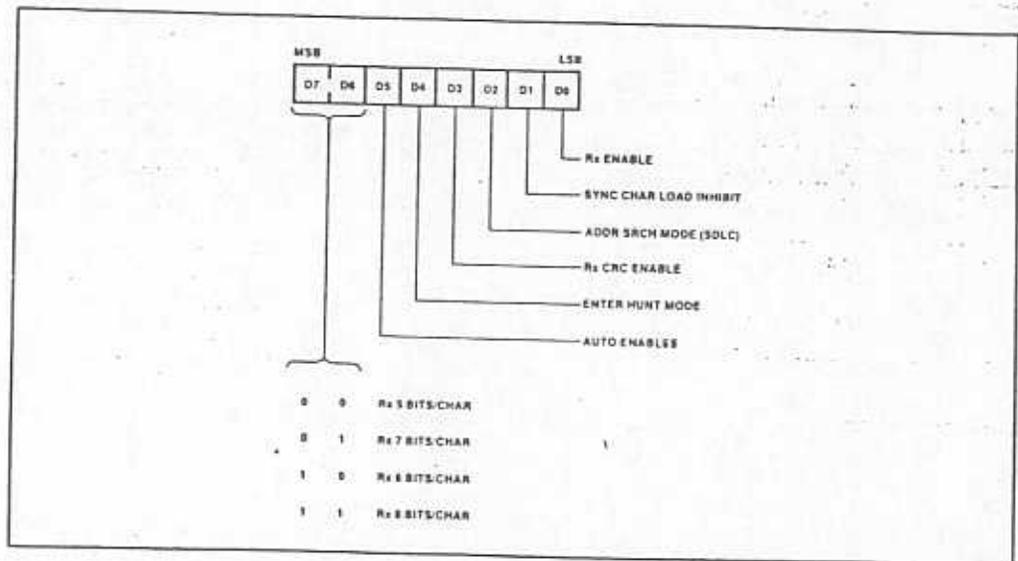
Write Register 2 (WR2): Channel B



WR2 CHANNEL B

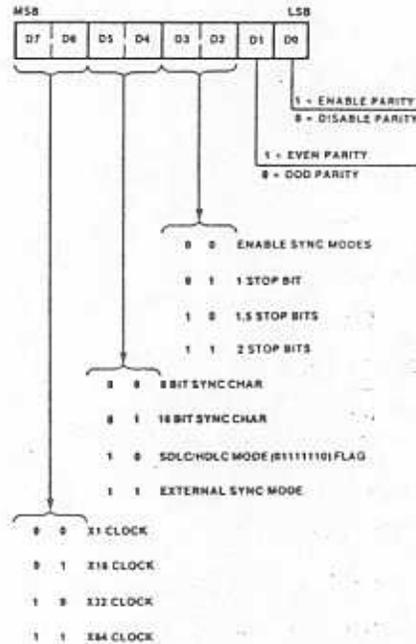
D7-D0 Interrupt vector—This register contains the value of the interrupt vector placed on the data bus during interrupt acknowledge sequences.

Write Register 3 (WR3):



- WR3**
- D0** Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
- D1** Sync Character Load Inhibit—A one prevents the receiver from loading sync characters into the receive buffers. In SDLC, this bit must be zero.
- D2** Address Search Mode—If the SDLC mode has been selected, the MPSC will receive all frames unless this bit is a 1. If this bit is a 1, the MPSC will receive only frames with address bytes that match the global address (0FFH) or the value loaded into WR6. This bit must be zero in non-SDLC modes.
- D3** Receive CRC Enable—A one in this bit enables (or re-enables) CRC calculation. CRC calculation starts with the last character placed in the Receiver FIFO. A zero in this bit disables, but does not reset, the Receiver CRC generator.
- D4** Enter Hunt Phase—After initialization, the MPSC automatically enters the Hunt mode. If synchronization is lost, the Hunt phase can be re-entered by writing a one to this bit.
- D5** Auto Enable—A one written to this bit causes  $\overline{CD}$  to be automatic enable signal for the receiver and  $\overline{CTS}$  to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of  $\overline{CD}$  and  $\overline{CTS}$  signals to setting/resetting their corresponding bits in the status register (RR0).
- D7, D6** Receive Character length
- 0 0 Receive 5 Data bits/character
  - 0 1 Receive 7 Data bits/character
  - 1 0 Receive 6 Data bits/character
  - 1 1 Receive 8 Data bits/character

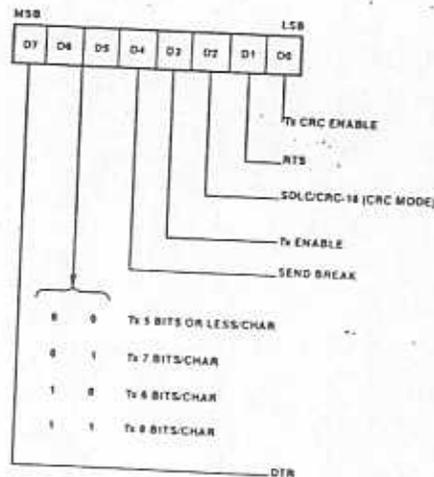
**Write Register 4 (WR4):**



- WR4**
- D0** Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.
- D1** Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and a zero causes it to send and expect odd parity.
- D3, D2** Stop bits/sync mode

- 0 0 Selects synchronous modes.
- 0 1 Async mode, 1 stop bit/character
- 1 0 Async mode, 1-1/2 stop bits/character
- 1 1 Async mode, 2 stop bits/character
- D5, D4 Sync mode select
  - 0 0 8 bit sync character
  - 0 1 16 bit sync character
  - 1 0 SDLC mode (Flag sync)
  - 1 1 External sync mode
- D7, D6 Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. 1x mode must be selected for synchronous modes. If the 1x mode is selected, bit synchronization must be done externally.
  - 0 0 Clock rate = Data rate x 1
  - 0 1 Clock rate = Data rate x 16
  - 1 0 Clock rate = Data rate x 32
  - 1 1 Clock rate = Data rate x 64

Write Register 5 (WR5):



WR5

- D0 Transmit CRC Enable—a one in this bit enables the transmitter CRC generator. The CRC calculation is done when a character is moved from the transmit buffer into the shift register. A zero in this bit disables CRC calculations. If this bit is not set when a transmitter underrun occurs, the CRC will not be sent.
- D1 Request to Send—a one in this bit forces the RTS pin active (low) and zero in this bit forces the RTS pin inactive (high).
- D2 CRC Select—a one in this bit selects the CRC - 16 polynomial ( $X^{16} + X^{15} + X^2 + 1$ ) and a zero in this bit selects the CCITT-CRC polynomial ( $X^{16} + X^{12} + X^5 + 1$ ). In SDLC mode, CCITT-CRC must be selected.
- D3 Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.
- D4 Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.
- D6, D5 Transmit Character length
  - 0 0 Transmit 1 - 5 bits/character
  - 0 1 Transmit 7 bits/character
  - 1 0 Transmit 6 bits/character
  - 1 1 Transmit 8 bits/character

Bits to be sent must be right justified least significant bit first, eg:

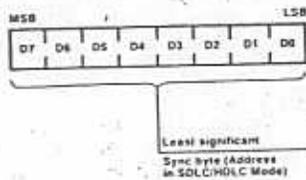
D7 D6 D5 D4 D3 D2 D1 D0  
 0 0 B5 B4 B3 B2 B1 B0

Five or less mode allows transmission of one to five bits per character. The microprocessor must format the data in the following way:

D7	D6	D5	D4	D3	D2	D1	D0		
1	1	1	1	0	0	0	0	B0	Sends one data bit
1	1	1	0	0	0	0	0	B1 B0	Sends two data bits
1	1	0	0	0	0	0	0	B2 B1 B0	Sends three data bits
1	0	0	0	0	0	0	0	B3 B2 B1 B0	Sends four data bits
0	0	0	0	0	0	0	0	B4 B3 B2 B1 B0	Sends five data bits

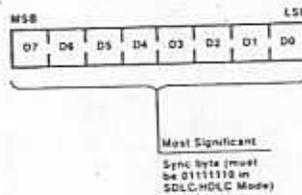
**D7** Data Terminal Ready—when set, this bit forces the  $\overline{\text{DTR}}$  pin active (low). When reset, this bit forces the  $\overline{\text{DTR}}$  pin inactive (high).

**Write Register 6 (WR6):**



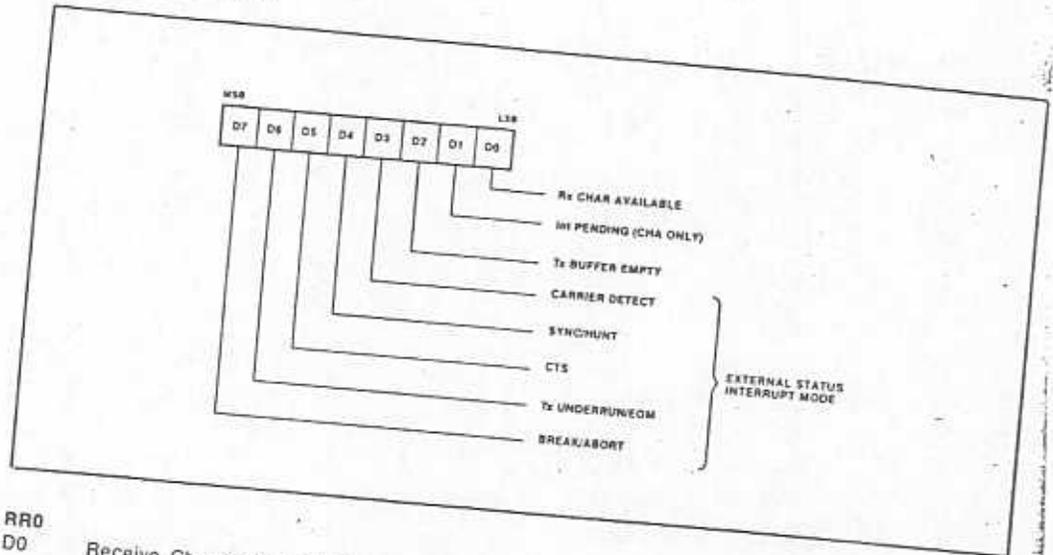
**WR6**  
**D7-D0** Sync/Address—this register contains the transmit sync character in Monosync mode, the low order 8 sync bits in Bisync mode, or the Address byte in SDLC mode.

**Write Register 7 (WR7):**



**WR7**  
**D7-D0** Sync/Flag—this register contains the receive sync character in Monosync mode, the high order 8 sync bits in Bisync mode, or the Flag character (01111110) in SDLC mode. WR7 is not used in External Sync mode.

Read Register 0 (RR0):



RR0

D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending\*—This Interrupt-Pending bit is reset when an EOI command is issued and there is no other interrupt request pending at that time.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the  $\overline{CD}$  pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the  $\overline{CD}$  pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the  $\overline{CD}$  pin immediately following a Reset External/Status Interrupt command.

\*In vector mode this bit is set at the falling edge of the second  $\overline{INTA}$  in an  $\overline{INTA}$  cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of  $\overline{RD}$  input after pointer 2 is specified. This bit is always zero in Channel B.

D4

Sync/Hunt—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that Sync/Hunt shows the state of the  $\overline{SYNDET}$  input. Any High-to-Low transition on the  $\overline{SYNDET}$  pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the  $\overline{SYNDET}$  pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the  $\overline{SYNDET}$  input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the  $\overline{SYNDET}$  input must be held High by the external logic until external character synchronization is achieved. A High at the  $\overline{SYNDET}$  input holds the Sync/Hunt status in the reset condition.

When external synchronization is achieved,  $\overline{\text{SYNDET}}$  must be driven Low on the second rising edge of  $\overline{\text{RxC}}$  after the rising edge of  $\overline{\text{RxC}}$  on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the  $\overline{\text{SYNDET}}$  input. Once  $\overline{\text{SYNDET}}$  is forced Low, it is good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. The High-to-Low transition of the  $\overline{\text{SYNDET}}$  output sets the Sync/Hunt bit, which sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt Command.

When the  $\overline{\text{SYNDET}}$  input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the MPSC again looks for a High-to-Low transition on the  $\overline{\text{SYNDET}}$  input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the MPSC is waiting for  $\overline{\text{SYNDET}}$  to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the MPSC establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the MPSC to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status Interrupt, which must also be cleared by the Reset External/Status Interrupt Command. Note that the  $\overline{\text{SYNDET}}$  pin acts as an output in this mode, and goes low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the MPSC. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The MPSC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

- D5 Clear to Send—this bit contains the inverted state of the CTS pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the CTS pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the CTS pin immediately following a Reset External/Status Interrupt command.
- D6 Transmitter Underrun/End of Message—this bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D<sub>6</sub> and D<sub>7</sub>). When the Transmit Underrun condition occurs, this bit is set, which causes the External/Status Interrupt which must be reset by issuing a Reset External/Status command (WR0, command 2).
- D7 Break/Abort—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

SDLC Residue Code Table (I Field Bits in 2 Previous Bytes)

RR1 D3, D2, D1	8 bits/char		7 bits/char		6 bits/char		5 bits/char	
	Previous Byte	2nd Prev. Byte						
1 0 0	0	3	0	2	0	1	0	5
0 1 0	0	4	0	3	0	2	0	1
1 1 0	0	5	0	4	0	3	0	2
0 0 1	0	6	0	5	0	4	0	3
1 0 1	0	7	0	6	0	5	—	—
0 1 1	0	8	0	—	0	—	—	—
1 1 1	1	8	—	—	—	—	—	—
0 0 0	2	8	1	7	0	6	0	4

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

D0

All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

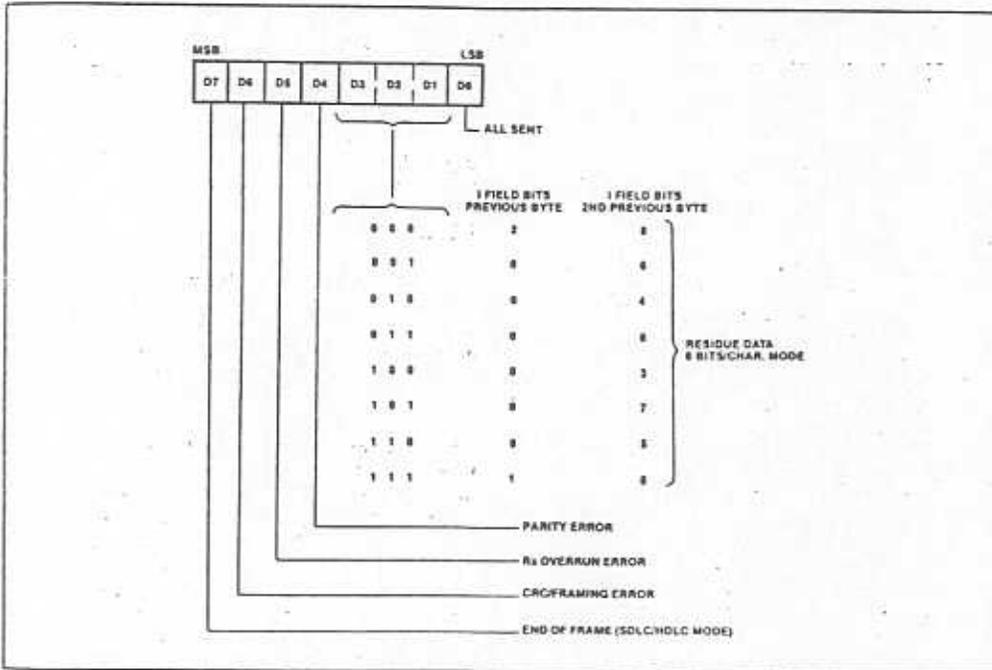
D3, D2, D1

Residue Codes—bit synchronous protocols allow I-fields that are not an integral number of characters. Since transfers from the MPSC to the CPU are character oriented, the residue codes provide the capability of receiving leftover bits. Residue bits are right justified in the last two data bytes received.

D4

Parity Error—If parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

Read Register 1 (RR1): (Special Receive Condition Mode)



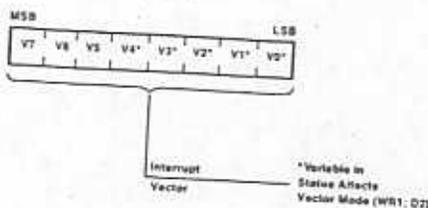
D5 Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the status affects vector mode, the overrun causes a special Receive Condition Vector.

D6 CRC/Framing Error—In async modes, a one in this bit indicates a receive fram-

ing error. In synchronous modes, a one in this bit indicates that the calculated CRC value does not match the last two bytes received. It can be reset by issuing an Error Reset command.

D7 End of Frame—this bit is valid only in SDLC mode. A one indicates that a valid ending flag has been received. This bit is reset either by an Error Reset command or upon reception of the first character of the next frame.

Read Register 2 (RR2):



**RR2 Channel B**  
**D7-D0** Interrupt vector—contains the interrupt vector programmed into WR2. If the status affects vector mode is selected (WR1; D2), it contains the modified vector (See WR2). RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

SYSTEM INTERFACE

General

The MPSC to Microprocessor System interface can be configured in many flexible ways. The basic interface types are polled, wait, interrupt driven, or direct memory access driven.

Polled operation is accomplished by repetitively reading the status of the MPSC, and making decisions based on that status. The MPSC can be polled at any time.

Wait operation allows slightly faster data throughput for the MPSC by manipulating the Ready input to the microprocessor. Block Read or Write Operations to the MPSC are started at will by the microprocessor and the MPSC deactivates its RDY signal if it is not yet ready to transmit the new byte, or if reception of new byte is not completed.

Interrupt driven operation is accomplished via an internal or external interrupt controller. When the MPSC requires service, it sends an interrupt request signal to the microprocessor, which responds with an interrupt acknowledge signal. When the internal or external interrupt controller receives the acknowledge, it vectors the microprocessor to a service routine, in which the transaction occurs.

DMA operation is accomplished via an external DMA controller. When the MPSC needs a data transfer, request a DMA cycle from the DMA controller. The DMA controller then takes control of the bus and simultaneously does a read from the MPSC and write to memory or vice-versa.

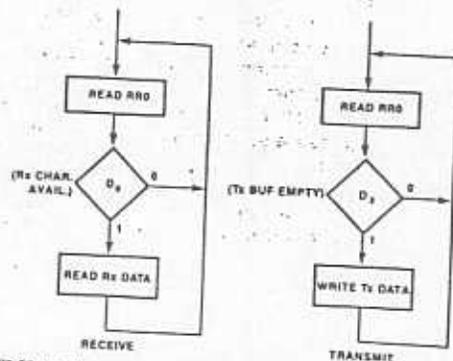
The following section describes the many configurations of these basic types of system interface techniques for both serial channels.

POLLED OPERATION:

In the polled mode, the CPU must monitor the desired conditions within the MPSC by reading the appropriate bits in the read registers. All data available, status, and error conditions are represented by the appropriate bits in read registers 0 and 1 for channels A and B.

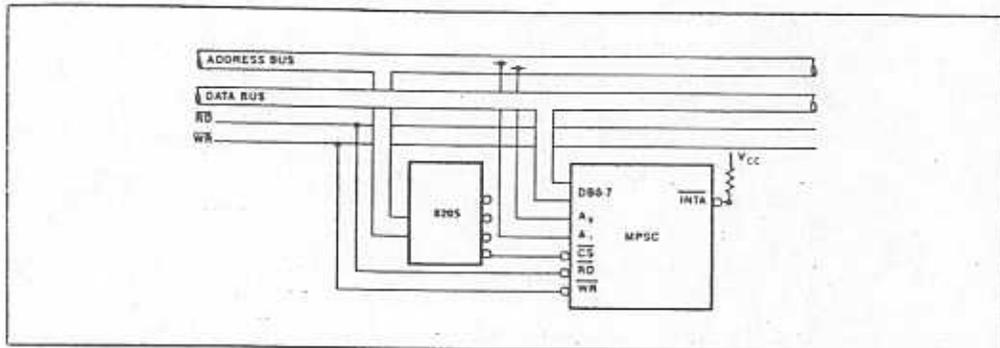
There are two ways in which the software task of monitoring the status of the MPSC has been reduced. One is the "ORing" of all conditions into the Interrupt Pending bit. (RR0; D1 channel A only). This bit is set when the MPSC requires service, allowing the CPU to monitor one bit instead of four status registers. The other is available when the "status-affects-vector" mode is selected. By reading RR2 Channel B, the CPU can read a vector whose value will indicate that one or more of group of conditions has occurred, narrowing the field of possible conditions. See WR2 and RR2 in the Detailed Command Description section.

Software Flow, Polled Operation



RR0; D0 is reset automatically when the data is read.  
 RR0; D2 is reset automatically when the data is written.

## Hardware Configuration, Polled Operation

**WAIT OPERATION:**

Wait Operation is intended to facilitate data transmission or reception using block move operations. If a block of data is to be transmitted, for example, the CPU can execute a String I/O instruction to the MPSC. After writing the first byte, the CPU will attempt to write a second byte immediately as is the case of block move. The MPSC forces the RDY signal low which inserts wait states in the CPU's write cycle until the transmit buffer is ready to accept a new byte. At that time, the RDY signal is high allowing the CPU to finish the write cycle. The CPU then attempts the third write and the process is repeated.

Similar operation can be programmed for the receiver. During initialization, wait on transmit (WR1; D5 = 0) or wait on receive (WR1; D5 = 1) can be selected. The wait operation can be enabled/disabled by setting/resetting the Wait Enable Bit (WR1; D7).

**CAUTION:** ANY CONDITION THAT CAN CAUSE THE TRANSMITTER TO STOP (EG, CTS GOES INACTIVE) OR THE RECEIVER TO STOP (EG, RX DATA STOPS) WILL CAUSE THE MPSC TO HANG THE CPU UP IN WAIT STATES UNTIL RESET. EXTREME CARE SHOULD BE TAKEN WHEN USING THIS FEATURE.

**INTERRUPT DRIVEN OPERATION:**

The MPSC can be programmed into several interrupt modes; Non-Vectored, 8085 vectored, and 8088/86 vectored. In both vectored modes, multiple MPSC's can be daisy-chained.

In the vectored mode, the MPSC responds to an interrupt acknowledge sequence by placing a call

instruction (8085 mode) and interrupt vector (8085 and 8088/86 mode) on the data bus.

The MPSC can be programmed to cause an interrupt due to up to 14 conditions in each channel. The status of these interrupt conditions is contained in Read Registers 0 and 1. These 14 conditions are all directed to cause 3 different types of internal interrupt request for each channel: receive/interrupts, transmit interrupts and external/status interrupts (if enabled).

This results in up to 6 internal interrupt request signals. The priority of those signals can be programmed to one of two fixed modes:

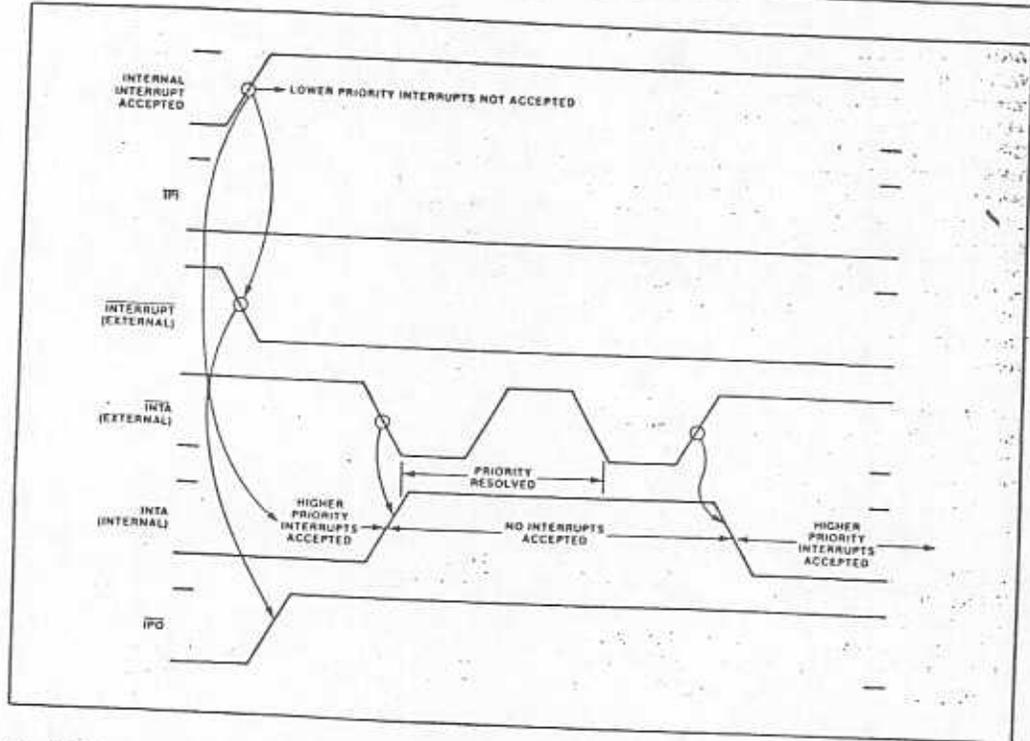
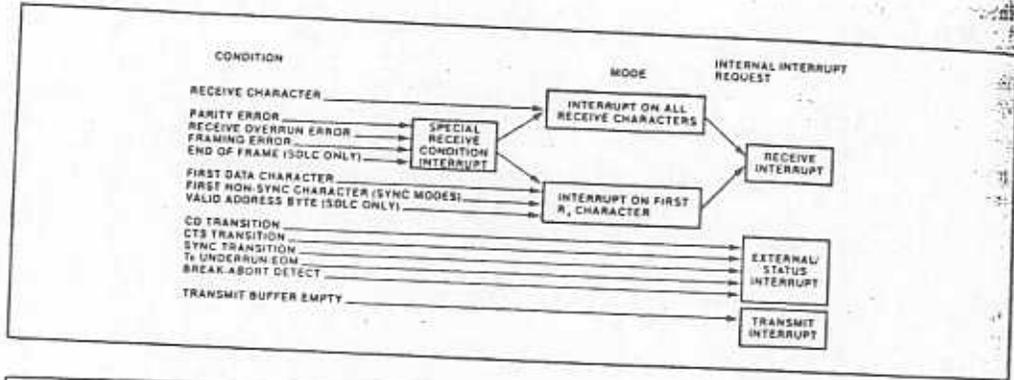
Highest Priority	Lowest Priority
RxA RxB TxA TxB ExTA ExTB	RxA TxA RxB TxB ExTA ExTB

The interrupt priority resolution works differently for vectored and non-vectored modes.

**PRIORITY RESOLUTION: VECTORED MODE**

Any interrupt condition can be accepted internally to the MPSC at any time, unless the MPSC's internal INTA signal is active, unless a higher priority interrupt is currently accepted, or if  $\overline{IP1}$  is inactive (high). The MPSC's internal INTA is set on the leading (falling) edge of the first External  $\overline{INTA}$  pulse and reset on the trailing (rising) edge of the second External  $\overline{INTA}$  pulse. After an interrupt is accepted internally, an External INT request is generated and the  $\overline{IPO}$  goes inactive.  $\overline{IPO}$  and  $\overline{IP1}$  are used for daisy-chaining MPSC's together.

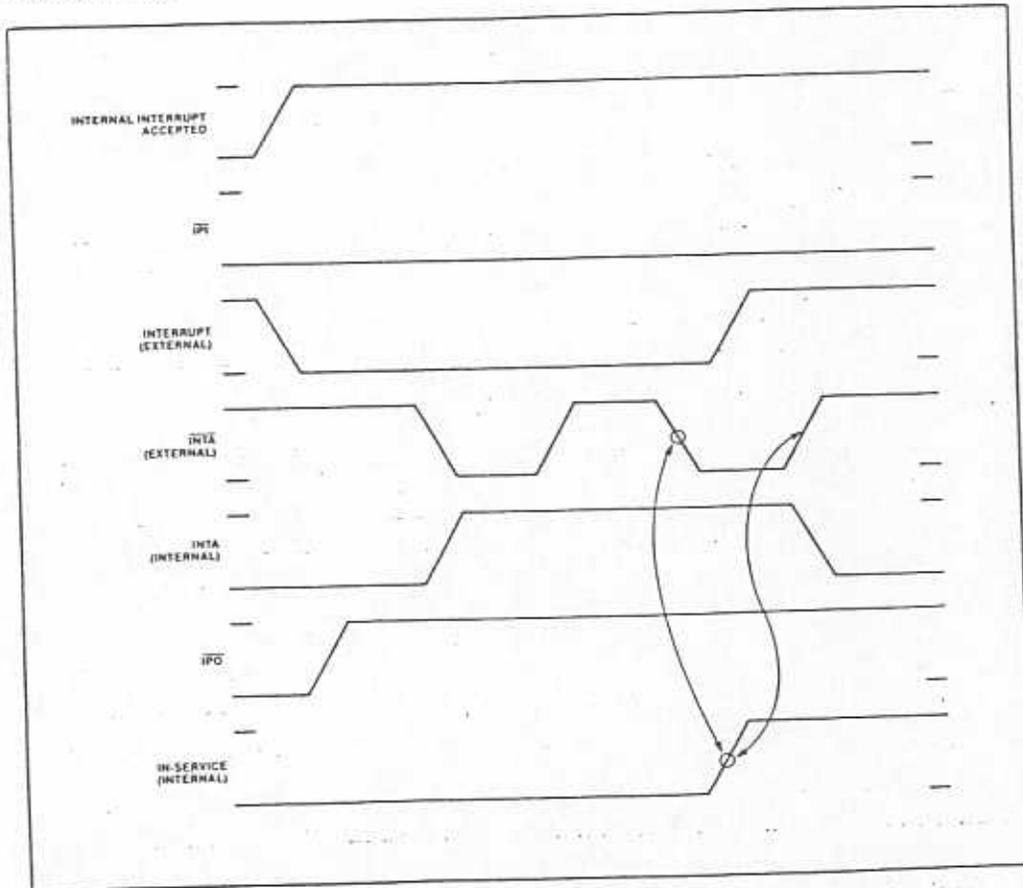
Interrupt Condition Grouping



The MPSC's internal INTA is set on the leading (falling) edge of the first external INTA pulse, and reset on the trailing (rising) edge of the second external INTA pulse. After an interrupt is accepted internally,

an external  $\overline{INT}$  request is generated and  $\overline{IPO}$  goes inactive (high).  $\overline{IPO}$  and  $\overline{IP}$  are used for daisy-chaining MPSC's together.

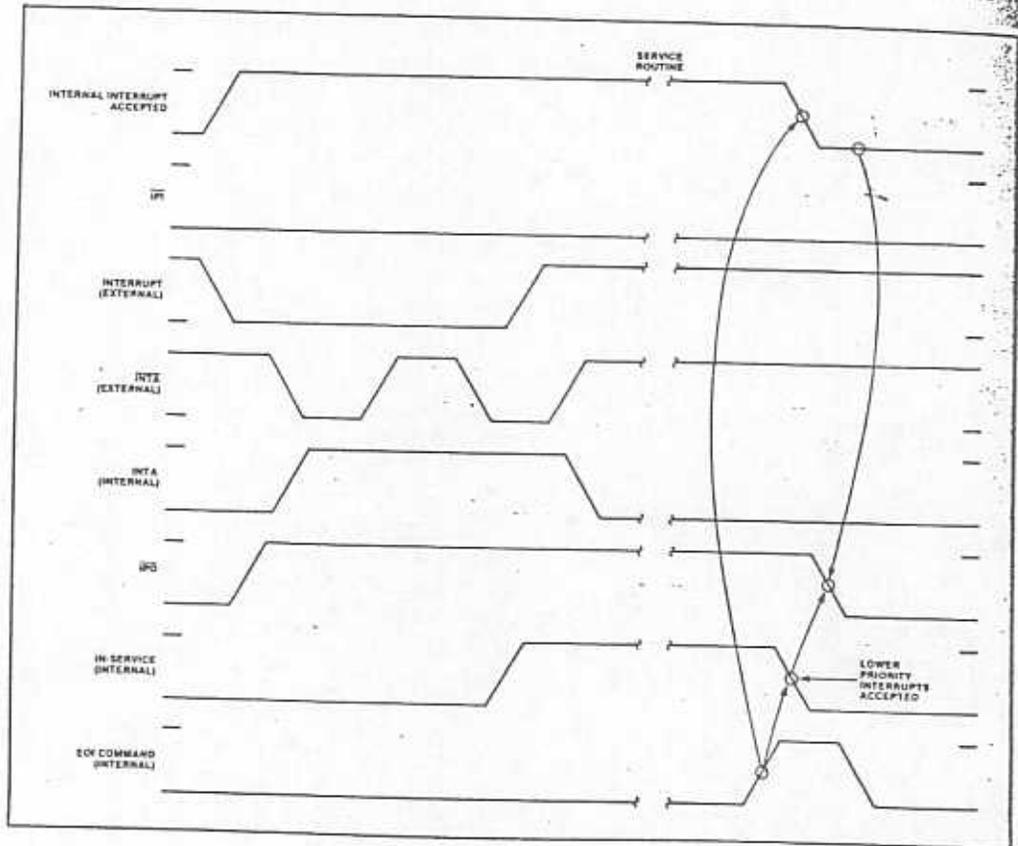
## In-Service Timing



Each of the six interrupt sources has an associated In-Service latch. After priority has been resolved, the

highest priority In-Service latch is set. After the In-Service latch is set, the INT pin goes inactive (high).

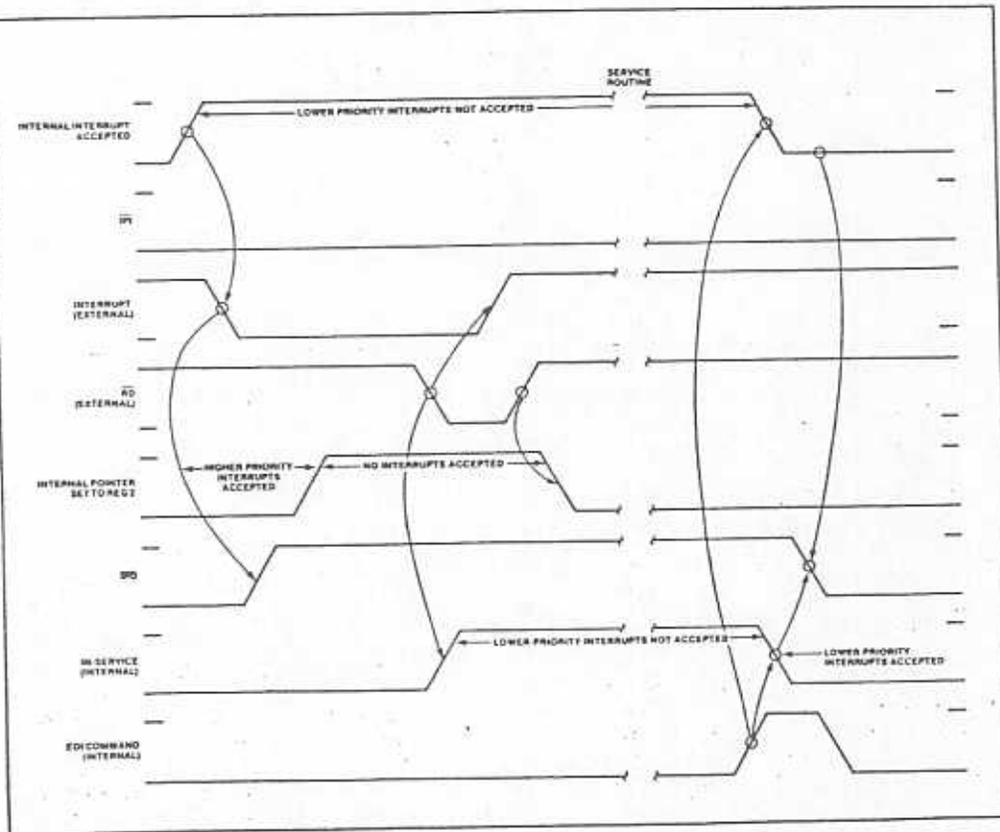
## EOI Command Timing



Lower priority interrupts are not accepted internally while the In-Service latch is set. However, higher priority interrupts are accepted internally and a new external INT request is generated. If the CPU responds with a new INTA sequence, the MPSC will respond as before, suspending the lower priority interrupt.

After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC. This command will cause an internal pulse that is used to reset the In-Service Latch which allows service for lower priority interrupts in the daisy-chain to resume, provided a new INTA sequence does not start for a higher priority interrupt (higher than the highest under service). If there is no interrupt pending internally, the IPI follows IPI.

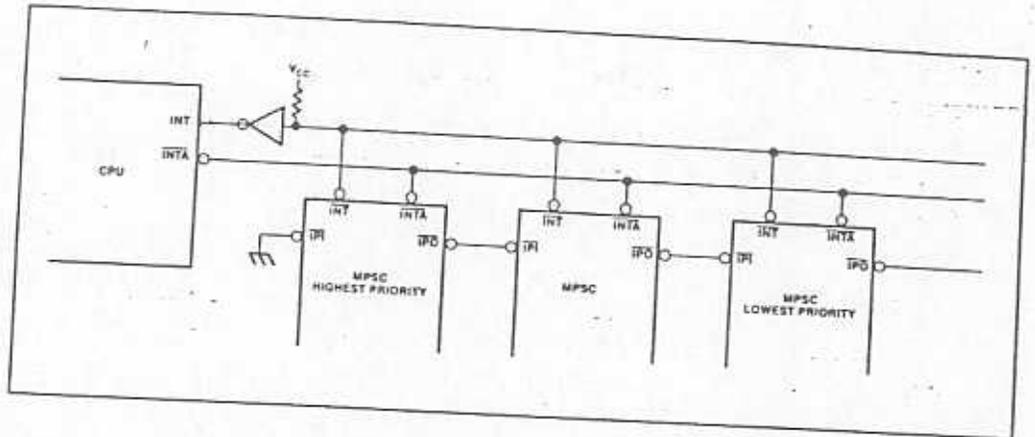
## Non-Vectored Interrupt Timing



### PRIORITY RESOLUTION: NON-VECTORED MODE

In non-vectored mode, the MPSC does not respond to interrupt acknowledge sequences. The INTA input (pin 27) must be pulled high for proper operation. The MPSC should be programmed to the Status-Affects-Vector mode, and the CPU should read RR2 (Ch. B) in its service routine to determine which interrupt requires service.

In this case, the internal pointer being set to RR2 provides the same function as the internal INTA signal in the vectored mode. It inhibits acceptance of any additional internal interrupts and its leading edge starts the interrupt priority resolution circuit. The interrupt priority resolution is ended by the leading edge of the read signal used by the CPU to retrieve the modified vector. The leading edge of read sets the In-Service latch and forces the external INT output inactive (high). The internal pointer is reset to zero after the trailing edge of the read pulse.



Note that if RR2 is specified but not read, no internal interrupts, regardless of priority, are accepted.

#### DAISY CHAINING MPSC:

In the vectored interrupt mode, multiple MPSC's can be daisy-chained on the same  $\overline{INT}$ ,  $\overline{INTA}$  signals. These signals, in conjunction with the  $\overline{IPI}$  and  $\overline{IPO}$  allow a daisy-chain-like interrupt resolution scheme. This scheme can be configured for either 8085 or 8086/88 based system.

In either mode, the same hardware configuration is called for. The  $\overline{INT}$  request lines are wire-OR'ed together at the input of a TTL inverter which drives the  $\overline{INT}$  pin of the CPU. The  $\overline{INTA}$  signal from the CPU drives all of the daisy-chained MPSC's.

The MPSC drives  $\overline{IPO}$  (Interrupt Priority Output) inactive (high) if  $\overline{IPI}$  (Interrupt Priority Input) is inactive (high), or if the MPSC has an interrupt pending.

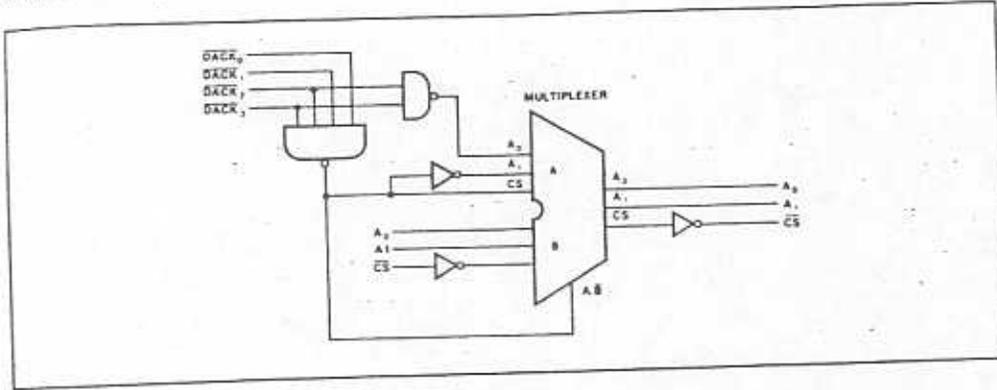
The  $\overline{IPO}$  of the highest priority MPSC is connected to the  $\overline{IPI}$  of the next highest priority MPSC, and so on.

If  $\overline{IPI}$  is active (low), the MPSC knows that all higher priority MPSC's have no interrupts pending. The  $\overline{IPI}$  pin of the highest priority MPSC is strapped active (low) to ensure that it always has priority over the rest.

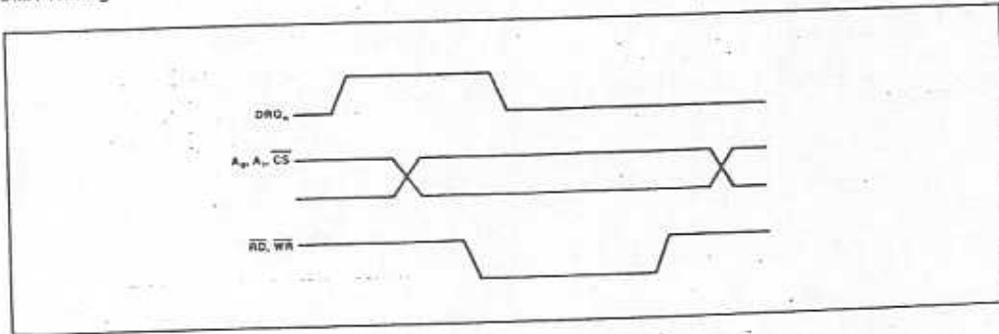
MPSC's Daisy-chained on an 8088/86 CPU should be programmed to the 8088/86 Interrupt mode (WR2; D4, D3 (Ch. A). MPSC's Daisy-chained on an 8085 CPU should be programmed to 8085 interrupt mode 1 if it is the highest priority MPSC. In this mode, the highest priority MPSC issues the CALL instruction during the first  $\overline{INTA}$  cycle, and the interrupting MPSC provides the interrupt vector during the following  $\overline{INTA}$  cycles. Lower priority MPSC's should be programmed to 8085 interrupt mode 2.

MPSC's used alone in 8085 systems should be programmed to 8085 mode 1 interrupt operation.

DMA Acknowledge Circuit



DMA Timing



DMA OPERATION

Each MPSC can be programmed to utilize up to four DMA channels: Transmit Channel A, Receive Channel A, Transmit Channel B, Receive Channel B. Each DMA Channel has an associated DMA Request line. Acknowledgement of a DMA cycle is done via normal data read or write cycles. This is accomplished by encoding the DACK signals to generate A<sub>0</sub>, A<sub>1</sub>, and CS signals, and multiplexing them with the normal A<sub>0</sub>, A<sub>1</sub>, and CS signals.

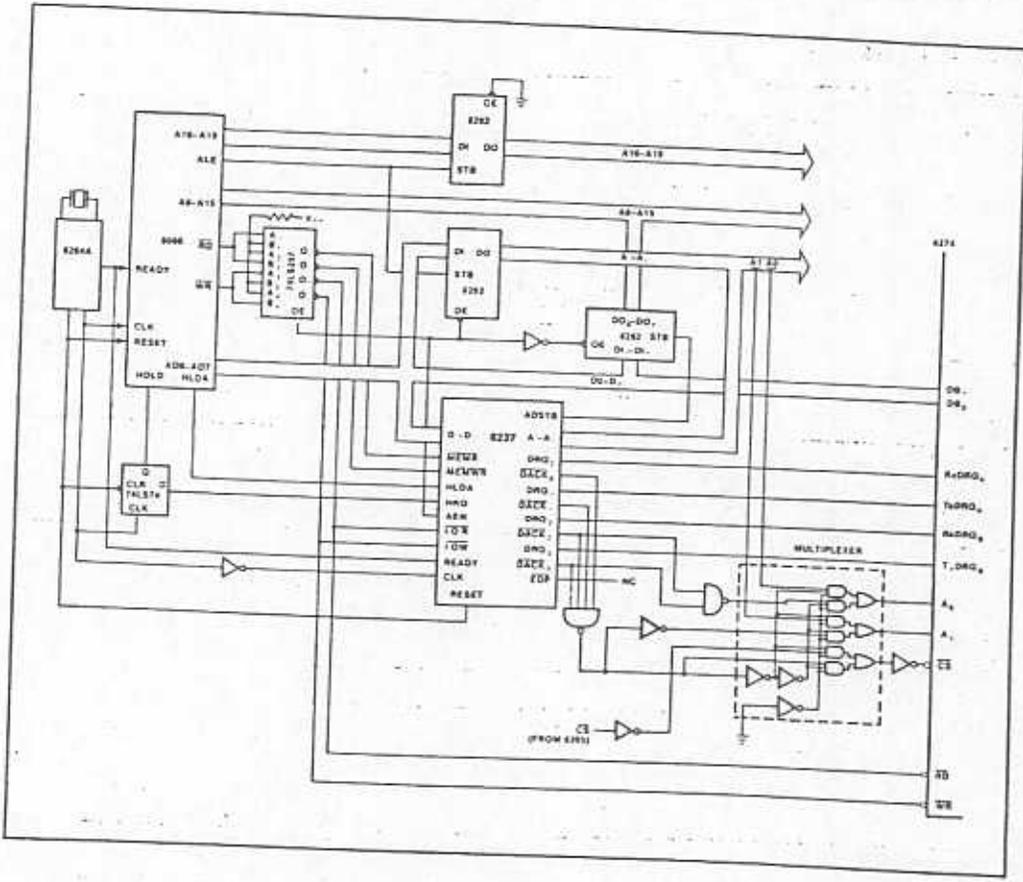
PERMUTATIONS

Channels A and B can be used with different system interface modes. In all cases it is impossible to poll the MPSC. The following table shows the possible

permutations of interrupt, wait, and DMA modes for channels A and B. Bits D<sub>1</sub>, D<sub>0</sub> of WR2 Ch. A determine these permutations.

Permutation WR2 Ch. A D <sub>1</sub> D <sub>0</sub>	Channel A	Channel B
0 0	Wait Interrupt Polled	Wait Interrupt Polled
0 1	DMA Polled	Interrupt Polled
1 0	DMA Polled	DMA Polled

D<sub>1</sub>, D<sub>0</sub> = 1, 1 is illegal.



## PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

### Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

### Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

### HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

### Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

### Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

### Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit under-run/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

### Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1=1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

### EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

### Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias .....	0°C to +70°C
Storage Temperature (Ceramic Package) .....	-65°C to +150°C
(Plastic Package) .....	-40°C to +125°C
Voltage On Any Pin With Respect to Ground .....	-0.5V to +7.0V

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = +5\text{V} \pm 10\%$ )

Symbol	Parameter	Min.	Max.	Units	Test Conditions
$V_{IL}$	Input Low Voltage	-0.5	+0.8	V	
$V_{IH}$	Input High Voltage	+2.0	$V_{CC} + 0.5$	V	
$V_{OL}$	Output Low Voltage		+0.45	V	$I_{OL} = 2.0\text{mA}$
$V_{OH}$	Output High Voltage	+2.4		V	$I_{OH} = -200\mu\text{A}$
$I_{IL}$	Input Leakage Current		$\pm 10$	$\mu\text{A}$	$V_{IN} = V_{CC}$ to $0\text{V}$
$I_{OL}$	Output Leakage Current		$\pm 10$	$\mu\text{A}$	$V_{OUT} = V_{CC}$ to $0\text{V}$
$I_{CC}$	$V_{CC}$ Supply Current		180	mA	

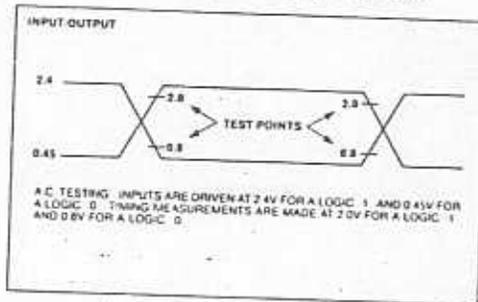
**CAPACITANCE** ( $T_A = 25^\circ\text{C}$ ;  $V_{CC} = \text{GND} = 0\text{V}$ )

Symbol	Parameter	Min.	Max.	Units	Test Conditions
$C_{IN}$	Input Capacitance		10	pF	$f_C = 1\text{MHz}$ ;
$C_{OUT}$	Output Capacitance		15	pF	Unmeasured
$C_{IO}$	Input/Output Capacitance		20	pF	pins returned to GND

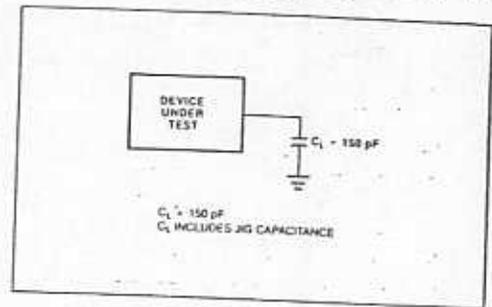
A.C. CHARACTERISTICS ( $T_s = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{cc} = +5V \pm 10\%$ )

Symbol	Parameter	Min.	Max.	Units	Test Conditions
$t_{CY}$	CLK Period	250	4000	ns	
$t_{CL}$	CLK Low Time	105	2000	ns	
$t_{CH}$	CLK High Time	105	2000	ns	
$t_r$	CLK Rise Time	0	30	ns	
$t_f$	CLK Fall Time	0	30	ns	
$t_{AR}$	A0, A1 Setup to $\overline{RD}_i$	0		ns	
$t_{AD}$	A0, A1 to Data Output Delay		200	ns	$C_L = 150$ pF
$t_{RA}$	A0, A1 Hold After $\overline{RD}_i$	0		ns	
$t_{RD}$	$\overline{RD}_i$ to Data Output Delay		200	ns	$C_L = 150$ pF
$t_{RR}$	$\overline{RD}$ Pulse Width	250		ns	
$t_{DF}$	Output Float Delay		120	ns	
$t_{AW}$	$\overline{CS}$ , A0, A1 Setup to $\overline{WR}_i$	0		ns	
$t_{WA}$	$\overline{CS}$ , A0, A1 Hold after $\overline{WR}_i$	0		ns	
$t_{WW}$	$\overline{WR}$ Pulse Width	250		ns	
$t_{DW}$	Data Setup to $\overline{WR}_i$		150	ns	
$t_{WD}$	Data Hold After $\overline{WR}_i$	0		ns	
$t_{PI}$	$\overline{IP}_i$ Setup to $\overline{INTA}_i$	0		ns	
$t_{IP}$	$\overline{IP}_i$ Hold after $\overline{INTA}_i$	10		ns	
$t_{II}$	$\overline{INTA}$ Pulse Width	250		ns	
$t_{IPO}$	$\overline{IP}_i$ to $\overline{IPO}$ Delay		100	ns	
$t_{ID}$	$\overline{INTA}_i$ to Data Output Delay		200	ns	
$t_{CO}$	$\overline{RD}$ or $\overline{WR}$ to $\overline{DRO}_i$		150	ns	
$t_{RV}$	Recovery Time Between Controls	300		ns	
$t_{CW}$	$\overline{CS}$ , A0, A1 to $\overline{RDY}_A$ or $\overline{RDY}_B$ Delay		140	ns	
$t_{DCY}$	Data Clock Cycle	4.5		tcy	
$t_{DCL}$	Data Clock Low Time	180		ns	
$t_{DCH}$	Data Clock High Time	180		ns	
$t_{TD}$	$\overline{TxC}$ to $\overline{TxD}$ Delay		300	ns	
$t_{PS}$	RxD Setup to $\overline{RxC}_i$	0		ns	
$t_{OH}$	RxD Hold after $\overline{RxC}_i$	140		ns	
$t_{ITD}$	$\overline{TxC}$ to $\overline{INT}$ Delay	4	6	tcy	
$t_{IRO}$	RxC to $\overline{INT}$ Delay	7	10	tcy	
$t_{PL}$	$\overline{CTS}$ , $\overline{CD}$ , $\overline{SYNDET}$ Low Time	200		ns	
$t_{PH}$	$\overline{CTS}$ , $\overline{CD}$ , $\overline{SYNDET}$ High Time	200		ns	
$t_{IPD}$	External $\overline{INT}$ from $\overline{CTS}$ , $\overline{CD}$ , $\overline{SYNDET}$		500	ns	

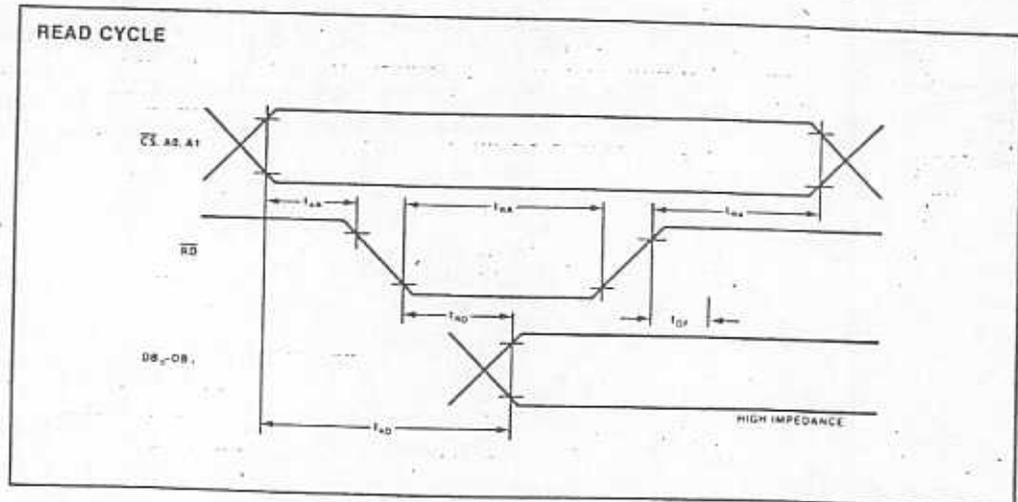
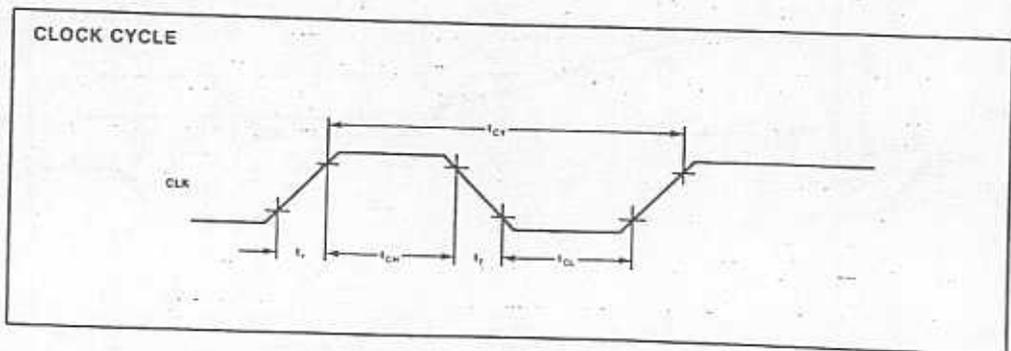
A.C. TESTING INPUT, OUTPUT WAVEFORM



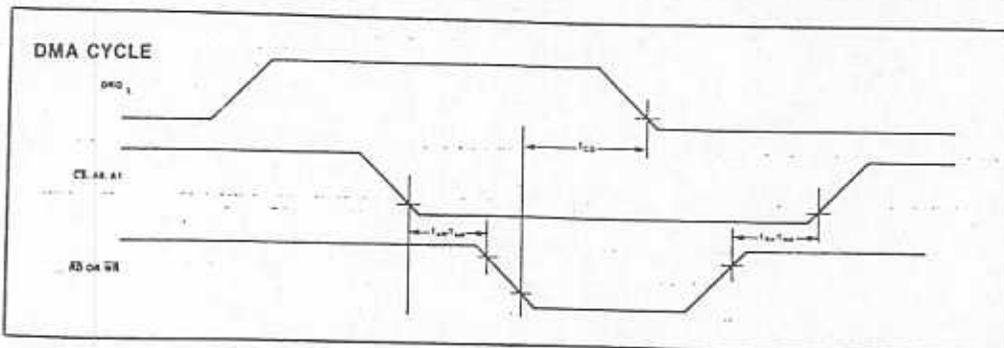
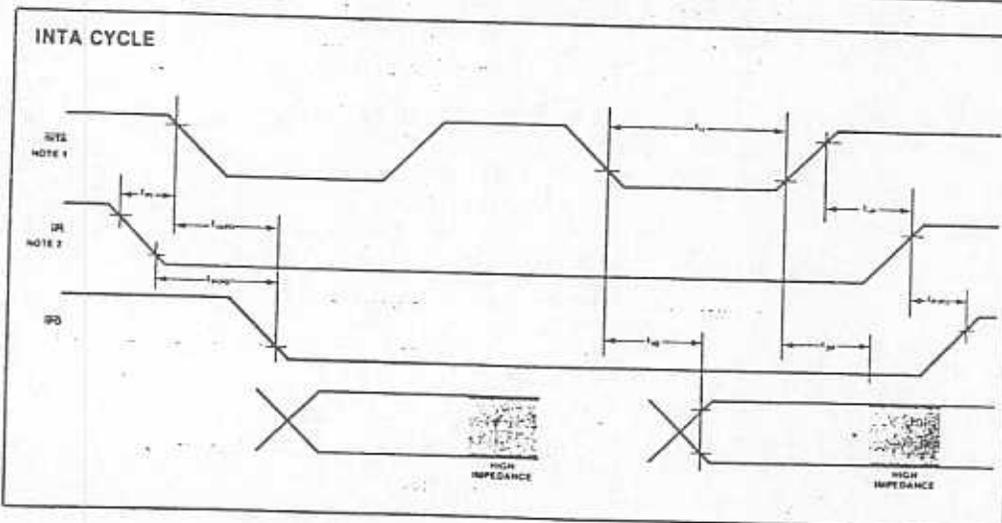
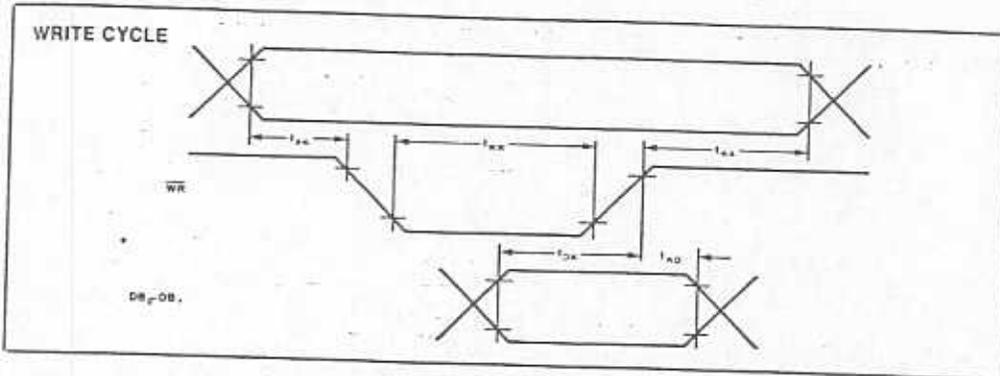
A.C. TESTING LOAD CIRCUIT



WAVEFORMS

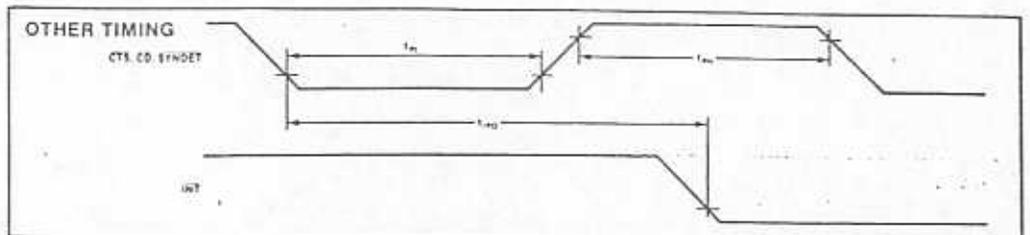
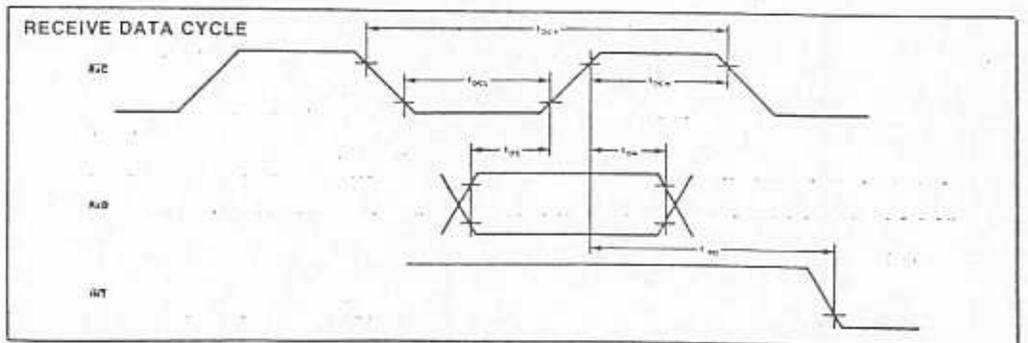
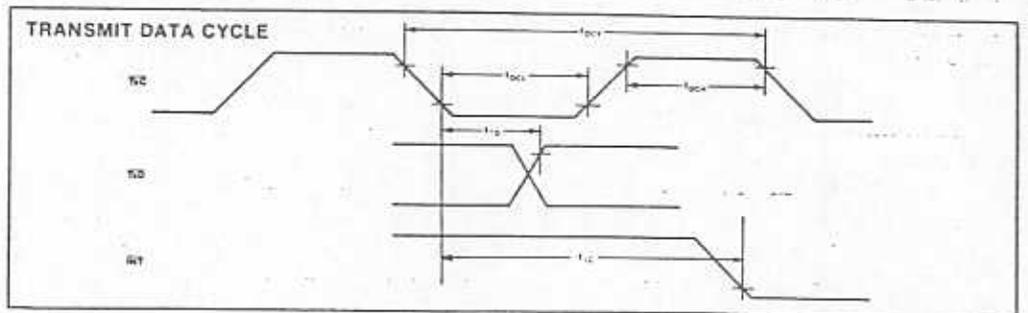
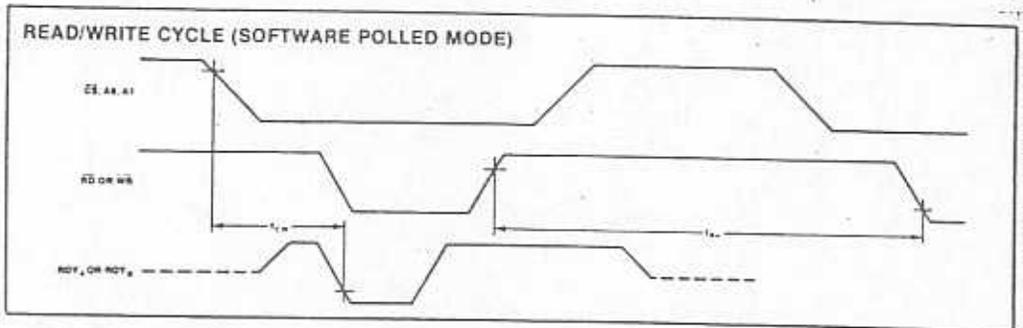


WAVEFORMS (Continued)



- NOTES:  
 1.  $\overline{INTA}$  signal acts as  $\overline{RD}$  signal.  
 2.  $\overline{IPI}$  signal acts as  $\overline{CS}$  signal.

WAVEFORMS (Continued)



1950

1950

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100

1950

8279

PROGRAMMABLE  
KEYBOARD/DISPLAY  
INTERFACE





## 8279/8279-5 PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

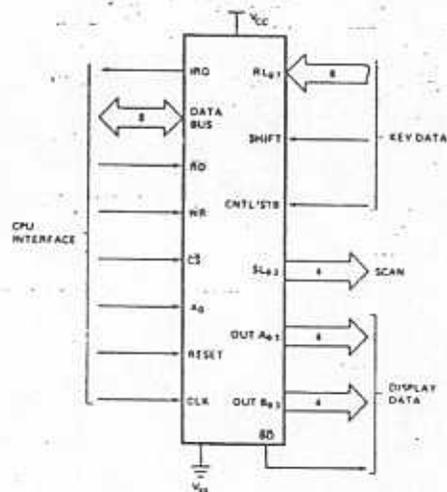


Figure 1. Logic Symbol



Figure 2. Pin Configuration

## HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Descriptions

Symbol	Pin No.	Name and Function
DB <sub>0</sub> -DB <sub>7</sub>	8	Bi-directional data bus: All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	1	Clock: Clock from system used to generate internal timing.
RESET	1	Reset: A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display—left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	1	Chip Select: A low on this pin enables the interface functions to receive or transmit.
A <sub>0</sub>	1	Buffer Address: A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
RD, WR	2	Input/Output Read and Write: These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	1	Interrupt Request: In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V <sub>ES</sub> , V <sub>CC</sub>	2	Ground and power supply pins.
SL <sub>0</sub> -SL <sub>3</sub>	4	Scan Lines: Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL <sub>0</sub> -RL <sub>7</sub>	8	Return Line: Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	1	Shift: The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	1	Control/Strobed Input Mode: For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode.  (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A <sub>0</sub> -OUT A <sub>3</sub> OUT B <sub>0</sub> -OUT B <sub>3</sub>	4 4	Outputs: These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL <sub>0</sub> -SL <sub>3</sub> ) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
BD	1	Blank Display: This output is used to blank the display during digit switching or by a display blanking command.

## FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

**Input Modes**

- Scanned Keyboard — with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

**Output Modes**

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ( $B_0 = D_0$ ,  $A_3 = D_7$ ).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

**PRINCIPLES OF OPERATION**

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

**I/O Control and Data Buffers**

The I/O control section uses the  $\overline{CS}$ ,  $A_0$ ,  $\overline{RD}$  and  $\overline{WR}$  lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by  $\overline{CS}$ . The character of the information, given or desired by the CPU, is identified by  $A_0$ . A logic one means the information is a command or status. A logic zero means the information is data.  $\overline{RD}$  and  $\overline{WR}$  determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ( $\overline{CS} = 1$ ), the devices are in a high impedance state. The drivers input during  $\overline{WR} \cdot \overline{CS}$  and output during  $\overline{RD} \cdot \overline{CS}$ .

**Control and Timing Registers and Timing Control**

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with  $A_0 = 1$  and then sending a  $\overline{WR}$ . The command is latched on the rising edge of  $\overline{WR}$ .

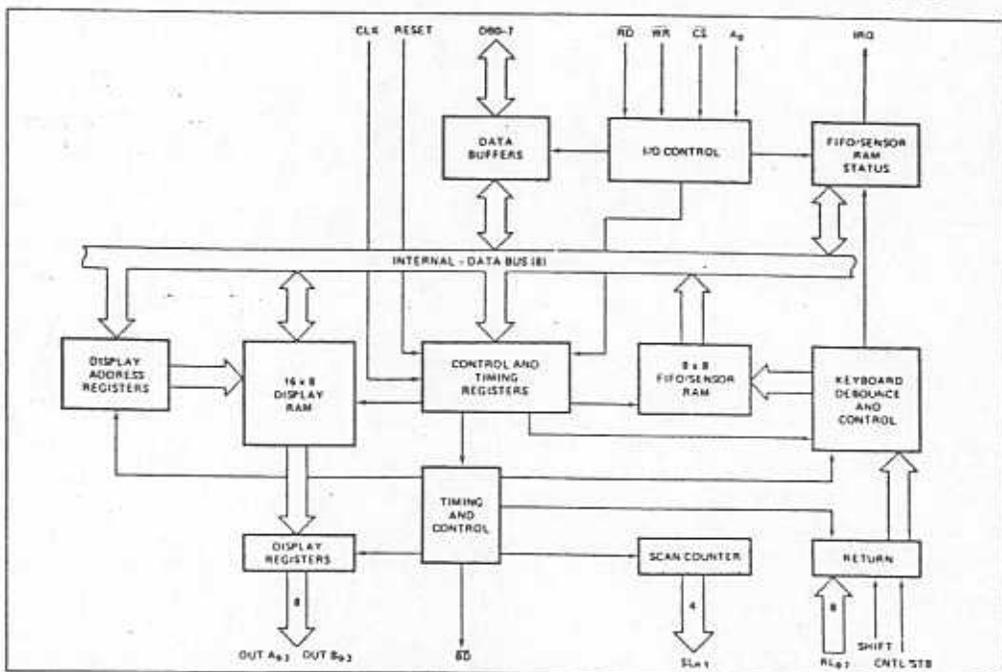


Figure 3. Internal Block Diagram

The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a  $\div N$  prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

#### Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note that when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

#### Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row. If the debounce circuit detects a closed switch, it waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

#### FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM. In Keyboard or Strobed Input modes, it is a FIFO. Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an RD with CS low and A<sub>0</sub> high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

#### Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

## SOFTWARE OPERATION

### 8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with CS low and A<sub>0</sub> high and are loaded to the 8279 on the rising edge of WR.

#### Keyboard/Display Mode Set

	MSB				LSB			
Code:	0	0	0	DD	DD	KK	KK	KK

Where DD is the Display Mode and KKK is the Keyboard Mode.

#### DD

- 0 0 8 8-bit character display — Left entry
- 0 1 16 8-bit character display — Left entry\*
- 1 0 8 8-bit character display — Right entry
- 1 1 16 8-bit character display — Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

#### KKK

- 0 0 0 Encoded Scan Keyboard — 2 Key Lockout\*
- 0 0 1 Decoded Scan Keyboard — 2-Key Lockout
- 0 1 0 Encoded Scan Keyboard — N-Key Rollover
- 0 1 1 Decoded Scan Keyboard — N-Key Rollover
- 1 0 0 Encoded Scan Sensor Matrix
- 1 0 1 Decoded Scan Sensor Matrix
- 1 1 0 Strobed Input, Encoded Display Scan
- 1 1 1 Strobed Input, Decoded Display Scan

#### Program Clock

Code:	0	0	1	PP	PP	PP	PP
-------	---	---	---	----	----	----	----

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits P P P P P determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, P P P P P should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

#### Read FIFO/Sensor RAM

Code:	0	1	0	AI	X	A	A	A
-------	---	---	---	----	---	---	---	---

X = Don't Care

The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-

\*Default after reset

board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read ( $A_0=0$ ) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set ( $AI=1$ ), each successive read will be from the subsequent row of the sensor RAM.

**Read Display RAM**

Code: 

0	1	1	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set ( $AI=1$ ), this row address will be incremented after each following read or write to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read or write address and the sense of the Auto-Increment mode for both operations.

**Write Display RAM**

Code: 

1	0	0	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with  $A_0=1$ , all subsequent writes with  $A_0=0$  will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

**Display Write Inhibit/Blanking**

Code: 

1	0	1	X	IW	IW	BL	BL
---	---	---	---	----	----	----	----

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag ( $IW=1$ ) for one of the ports, the port becomes masked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit  $B_0$  corresponds to bit  $D_0$  on the CPU bus, and that bit  $A_3$  corresponds to bit  $D_7$ .

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

**Clear**

Code: 

1	1	0	$C_D$	$C_D$	$C_D$	$C_F$	$C_A$
---	---	---	-------	-------	-------	-------	-------

The  $C_D$  bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

$C_D$	$C_D$	$C_D$	
0	X		All Zeros (X = Don't Care)
1	0		AB = Hex 20 (0010 0000)
1	1		All Ones

Enable clear display when = 1 (or by  $C_A=1$ )

During the time the Display RAM is being cleared (~160  $\mu$ s), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the  $C_F$  bit is asserted ( $C_F=1$ ), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

$C_A$ , the Clear All bit, has the combined effect of  $C_D$  and  $C_F$ ; it uses the  $C_D$  clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

**End Interrupt/Error Mode Set**

Code: 

1	1	1	E	X	X	X	X
---	---	---	---	---	---	---	---

 X = Don't care.

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset.)

For the N-key rollover mode — if the E bit is programmed to "1" the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

**Status Word**

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when  $A_0$  is high and  $\overline{CS}$  and  $\overline{RD}$  are low. See Interface Considerations for more detail on status word.

**Data Read**

Data is read when  $A_0$ ,  $\overline{CS}$  and  $\overline{RD}$  are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of  $\overline{RD}$  will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

**Data Write**

Data that is written with  $A_0$ ,  $\overline{CS}$  and  $\overline{WR}$  low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of  $\overline{WR}$  occurs if AI set by the latest display command.

## INTERFACE CONSIDERATIONS

### Scanned Keyboard Mode, 2-Key Lockout

There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRO will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

### Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is, the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

### Scanned Keyboard — Special Error Modes

For N-key rollover mode the user can program a special error mode. This is done by the "End Interrupt/Error Mode Set" command. The debounce cycle and key-validity check are as in normal N-key mode. If during a single debounce cycle, two keys are found depressed, this is considered a simultaneous multiple depression, and sets an error flag. This flag will prevent any further writing into the FIFO and will set interrupt (if not yet set). The error flag could be read in this mode by reading the FIFO STATUS word. (See "FIFO STATUS" for further details.) The error flag is reset by sending the normal CLEAR command with  $Cr = 1$ .

### Sensor Matrix Mode

In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is inputted directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them.

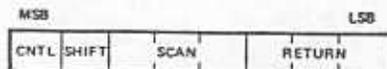
The IRO line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRO line is cleared by the first data read operation if the Auto-

Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

Note: Multiple changes in the matrix Addressed by (SL<sub>0-3</sub> = 0) may cause multiple interrupts. (SL<sub>0</sub> = 0 in the Decoded Mode). Reset may cause the 8279 to see multiple changes.

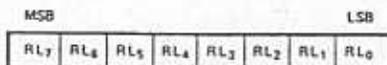
### Data Format

In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

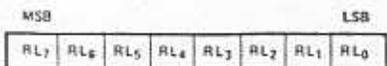


SCANNED KEYBOARD DATA FORMAT

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.



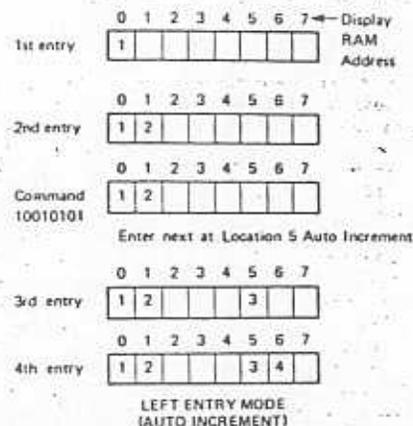
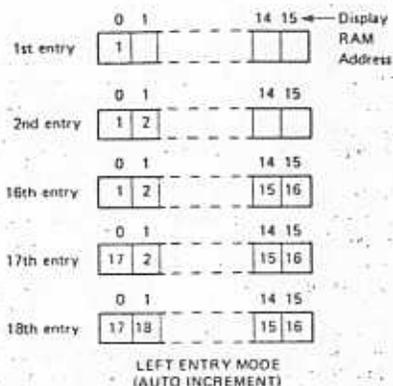
In Strobed Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input.



### Display

#### Left Entry

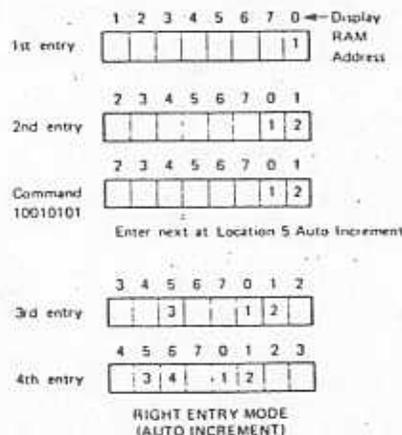
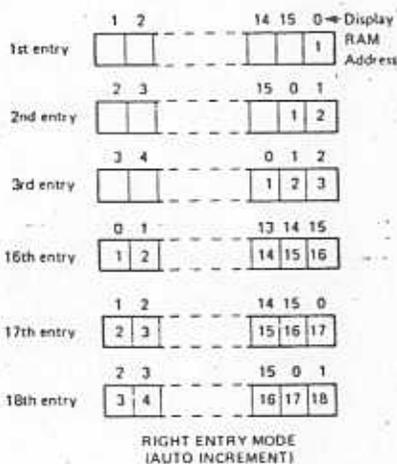
Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.



**Right Entry**

Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.

In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted:

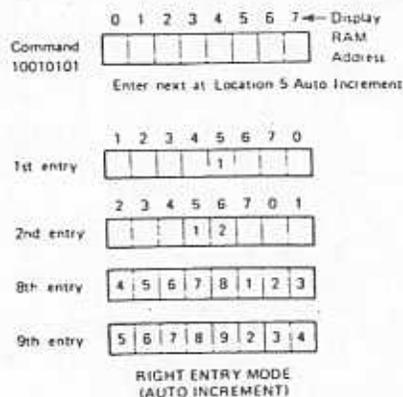


Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

Starting at an arbitrary location operates as shown below:

**Auto Increment**

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



Entry appears to be from the initial entry point.

**8/16 Character Display Formats**

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

**G. FIFO Status**

FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All Data command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.

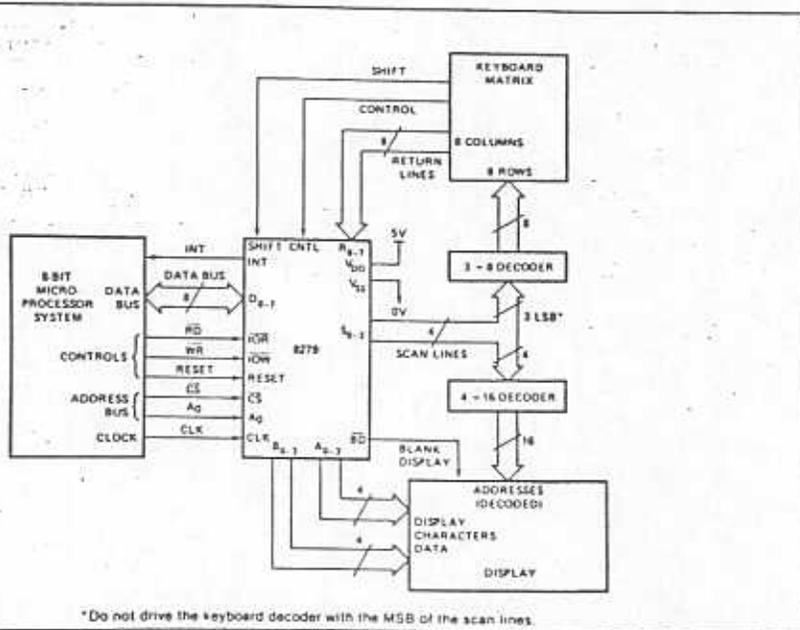
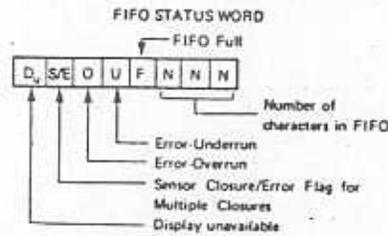


Figure 4. System Block Diagram

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature .....	0°C to 70°C
Storage Temperature .....	-65°C to 125°C
Voltage on any Pin with Respect to Ground .....	-0.5V to +7V
Power Dissipation .....	1 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**D.C. CHARACTERISTICS** [ $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{SS} = 0\text{V}$ , (NOTE 3)]\*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
$V_{IL1}$	Input Low Voltage for Return Lines	-0.5	1.4	V	
$V_{IL2}$	Input Low Voltage for All Others	-0.5	0.8	V	
$V_{IH1}$	Input High Voltage for Return Lines	2.2		V	
$V_{IH2}$	Input High Voltage for All Others	2.0		V	
$V_{OL}$	Output Low Voltage		0.45	V	Note 1
$V_{OH1}$	Output High Voltage on Interrupt Line	3.5		V	Note 2
$V_{OH2}$	Other Outputs	2.4			$I_{OH} = -400 \mu\text{A}$ 8279-5 $-100 \mu\text{A}$ 8279
$I_{IL1}$	Input Current on Shift, Control and Return Lines		+10 -100	$\mu\text{A}$	$V_{IN} = V_{CC}$ $V_{IN} = 0\text{V}$
$I_{IL2}$	Input Leakage Current on All Others		$\pm 10$	$\mu\text{A}$	$V_{IN} = V_{CC}$ to 0V
$I_{OFL}$	Output Float Leakage		$\pm 10$	$\mu\text{A}$	$V_{OUT} = V_{CC}$ to 0.45V
$I_{CC}$	Power Supply Current		120	mA	

**CAPACITANCE**

Symbol	Parameter	Typ.	Max.	Unit	Test Conditions
$C_{IN}$	Input Capacitance	5	10	pF	$f_C = 1 \text{ MHz}$ Unmeasured pins returned to $V_{SS}$
$C_{OUT}$	Output Capacitance	10	20	pF	

**A.C. CHARACTERISTICS** [ $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{SS} = 0\text{V}$ , (Note 3)] \*

**Bus Parameters**

**READ CYCLE**

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{AR}$	Address Stable Before READ	50		0		ns
$t_{RA}$	Address Hold Time for READ	5		0		ns
$t_{RR}$	READ Pulse Width	420		250		ns
$t_{RD}^{(4)}$	Data Delay from READ		300		150	ns
$t_{AD}^{(4)}$	Address to Data Valid		450		250	ns
$t_{DF}$	READ to Data Floating	10	100	10	100	ns
$t_{RCY}$	Read Cycle Time	1		1		$\mu\text{s}$



8279/8279-5

## A.C. CHARACTERISTICS (Continued)

## WRITE CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{AW}$	Address Stable Before WRITE	50		0		ns
$t_{WA}$	Address Hold Time for WRITE	20		0		ns
$t_{WW}$	WRITE Pulse Width	400		250		ns
$t_{DW}$	Data Set Up Time for WRITE	300		150		ns
$t_{WD}$	Data Hold Time for WRITE	40		0		ns
$t_{WCY}$	Write Cycle Time	1		1		$\mu$ s

## OTHER TIMINGS

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{\phi W}$	Clock Pulse Width	230		120		nsec
$t_{CY}$	Clock Period	500		320		nsec

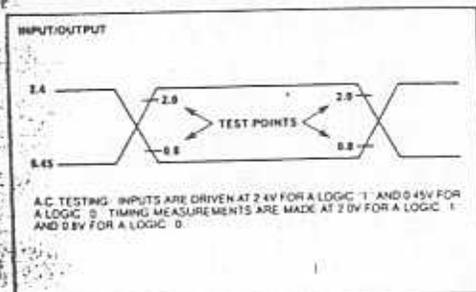
Keyboard Scan Time ..... 5.1 msec  
 Keyboard Debounce Time ..... 10.3 msec  
 Key Scan Time ..... 80  $\mu$ sec  
 Display Scan Time ..... 10.3 msec

Digit-on Time ..... 480  $\mu$ sec  
 Blanking Time ..... 160  $\mu$ sec  
 Internal Clock Cycle<sup>[5]</sup> ..... 10  $\mu$ sec

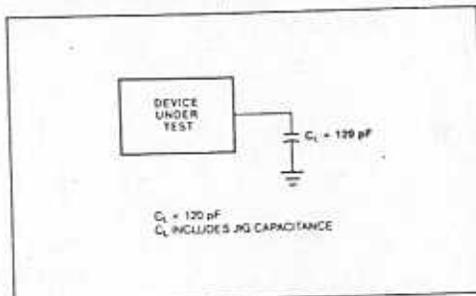
## NOTES:

1. 8279,  $I_{OL} = 1.6\text{mA}$ ; 8279-5,  $I_{OL} = 2.2\text{mA}$ .
  2.  $I_{OH} = -100\ \mu\text{A}$ .
  3. 8279,  $V_{CC} = +5V \pm 5\%$ ; 8279-5,  $V_{CC} = +5V \pm 10\%$ .
  4. 8279,  $C_L = 100\text{pF}$ ; 8279-5,  $C_L = 150\text{pF}$ .
  5. The Prescaler should be programmed to provide a 10  $\mu$ s internal clock cycle.
- \* For Extended Temperature EXPRESS, use M8279A electrical parameters.

## A.C. TESTING INPUT, OUTPUT WAVEFORM

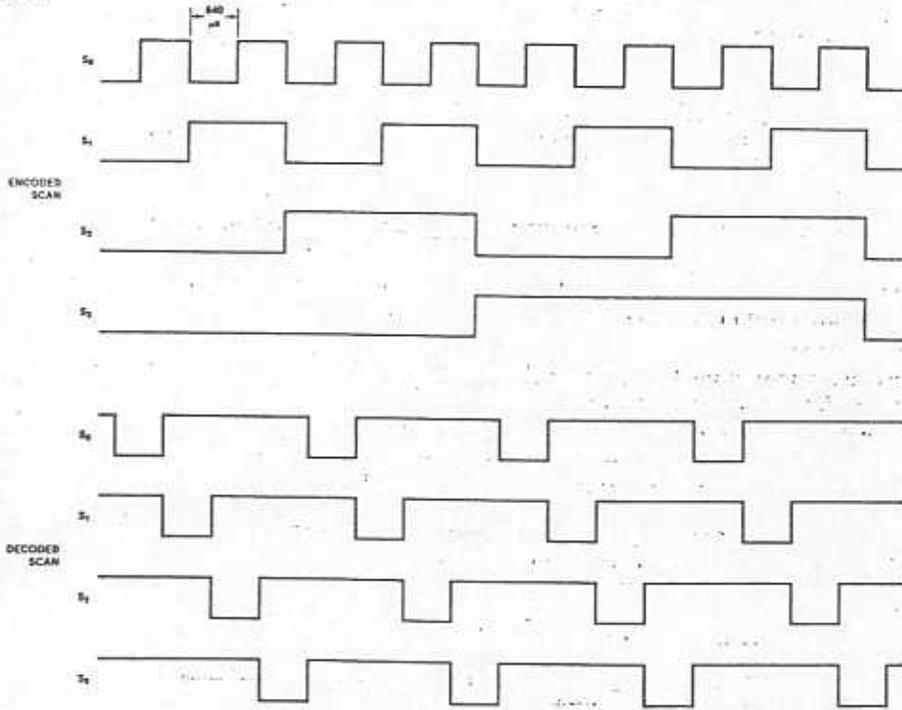


## A.C. TESTING LOAD CIRCUIT

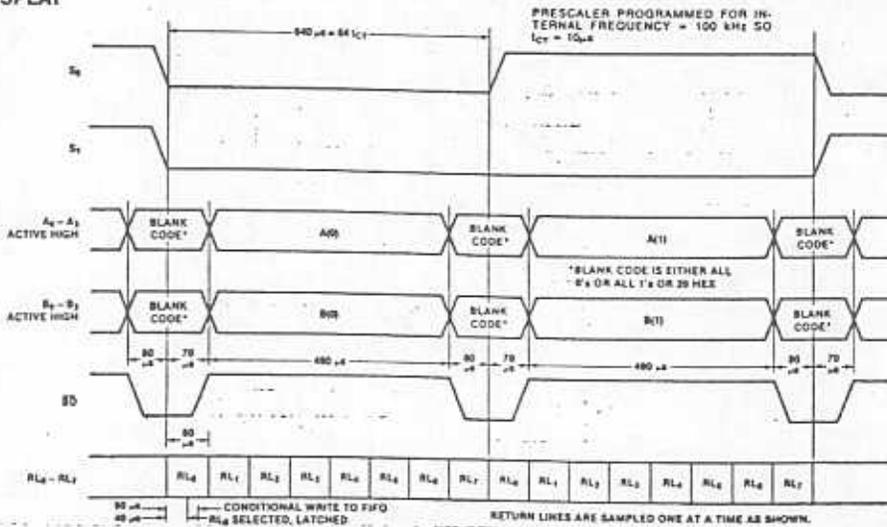


WAVEFORMS (Continued)

SCAN



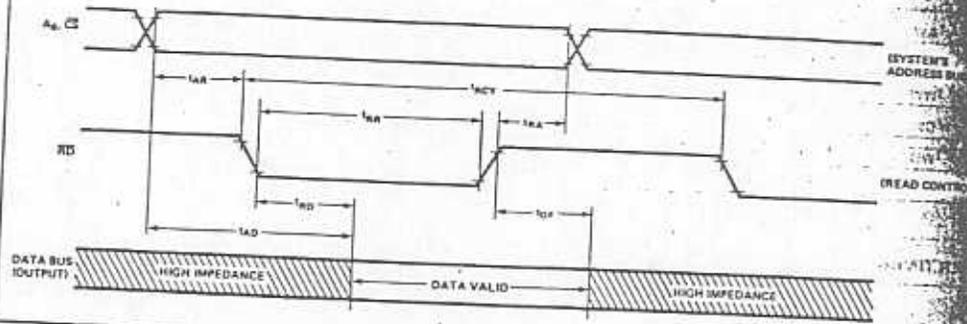
DISPLAY



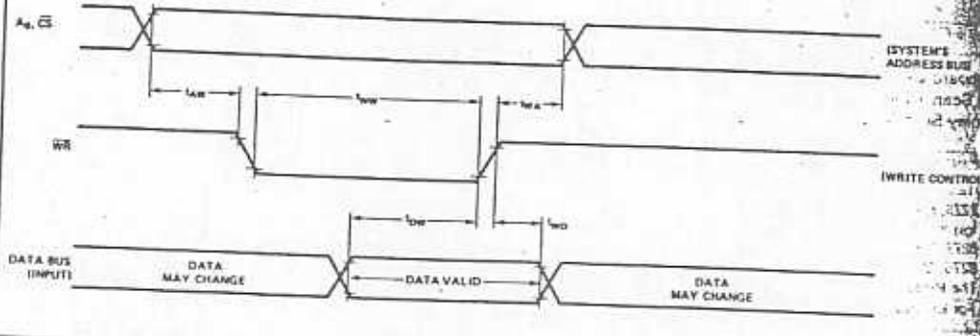
NOTE: SHOWN IS ENCODED SCAN LEFT ENTRY  
 S<sub>0</sub>-S<sub>3</sub> ARE NOT SHOWN BUT THEY ARE SIMPLY S<sub>0</sub>, DIVIDED BY 2 AND 4

WAVEFORMS

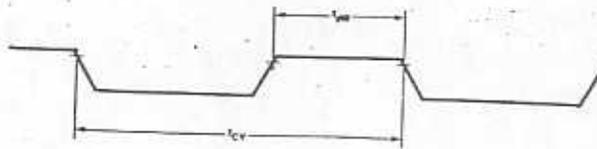
READ OPERATION



WRITE OPERATION



CLOCK INPUT



McClellan AFB  
Contract #: F04606-85-C0733  
CDRL Item: E00D  
15 May 90

**CHARTS AND FIGURES**

**FOR THE**

**AN/FMQ-13(V)**

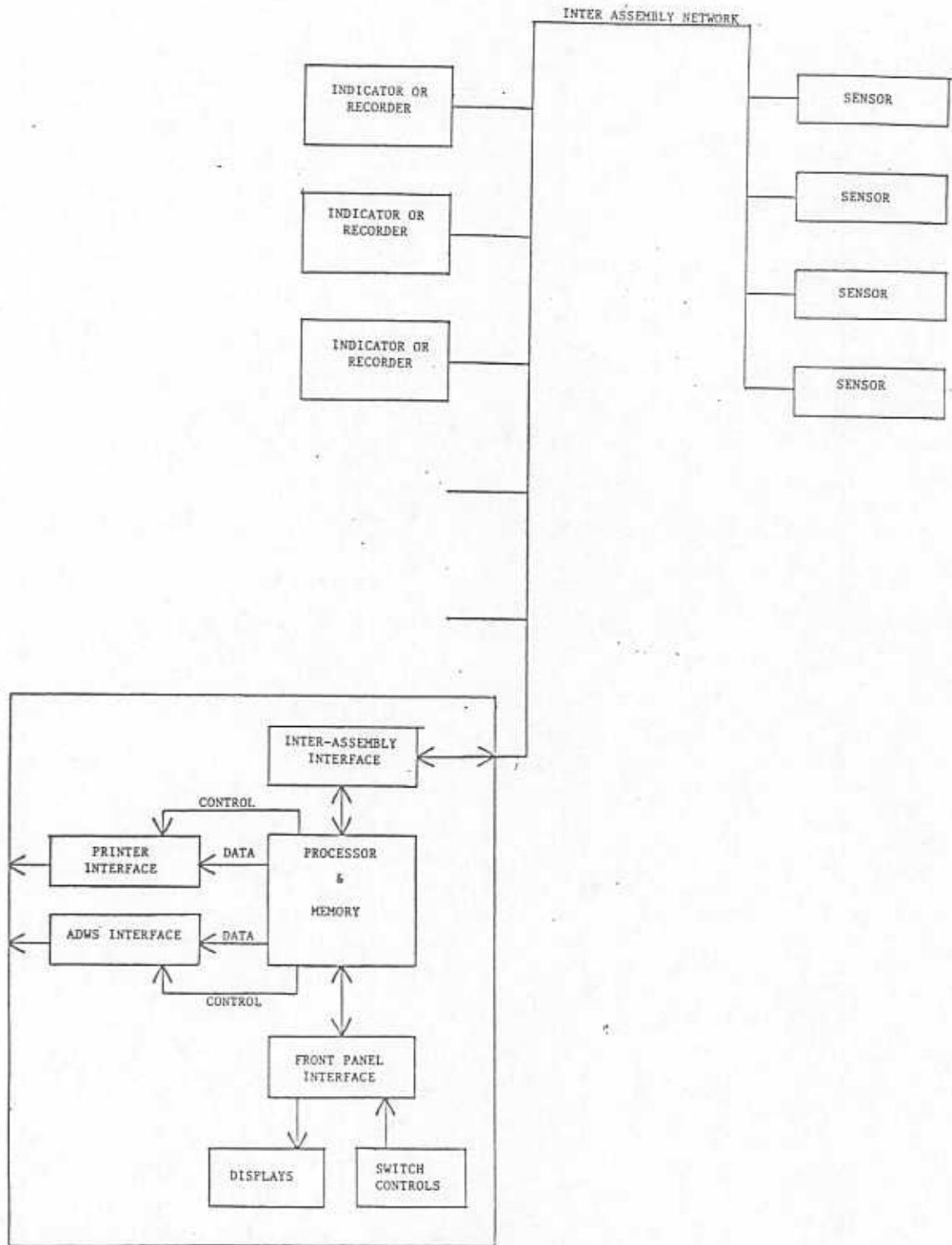
**DIGITAL INDICATOR/RECORDER**

**APPLICATION PROGRAM**

**(DWG # 8200-1010)**

Figure 1

SYSTEM BLOCK DIAGRAM



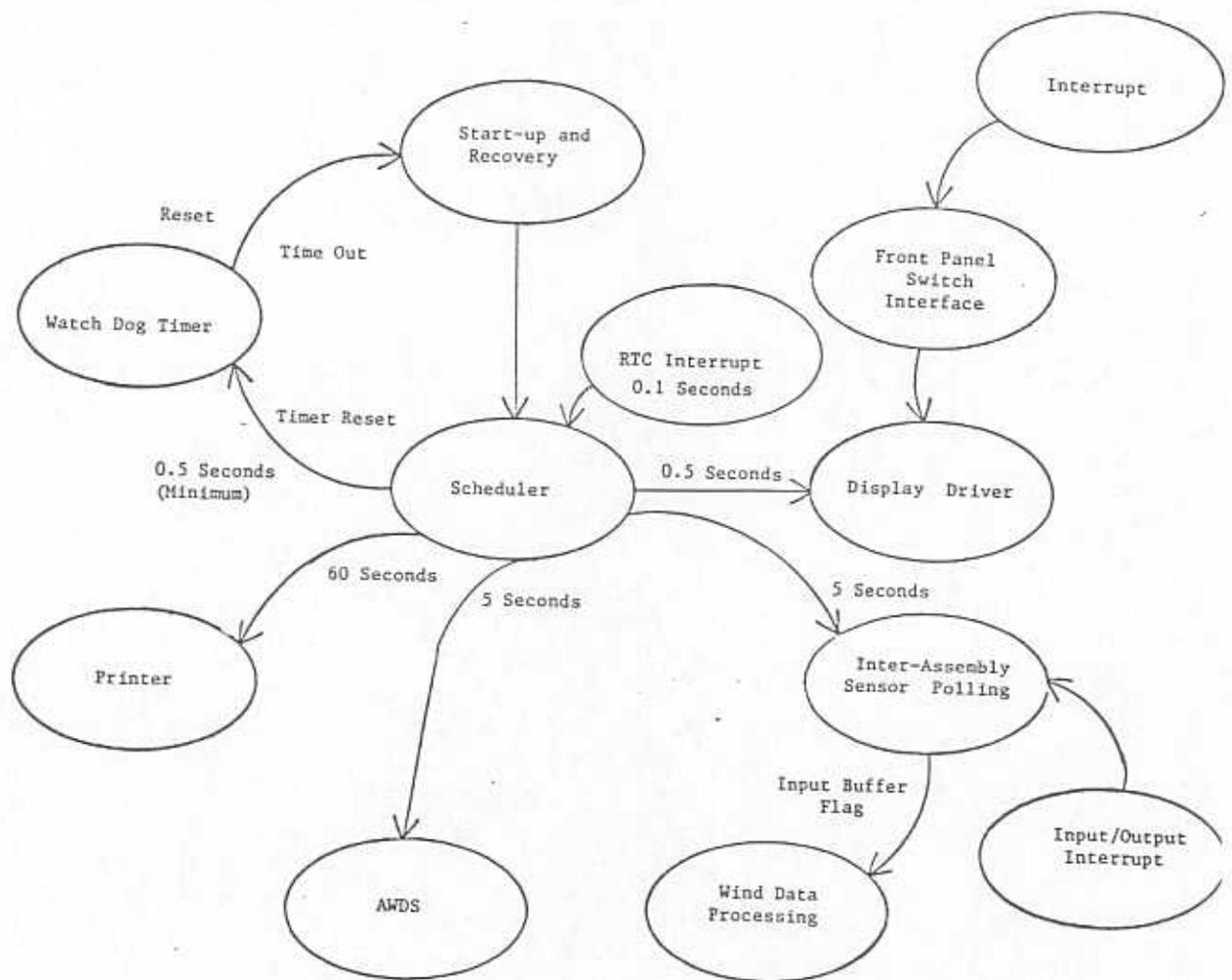
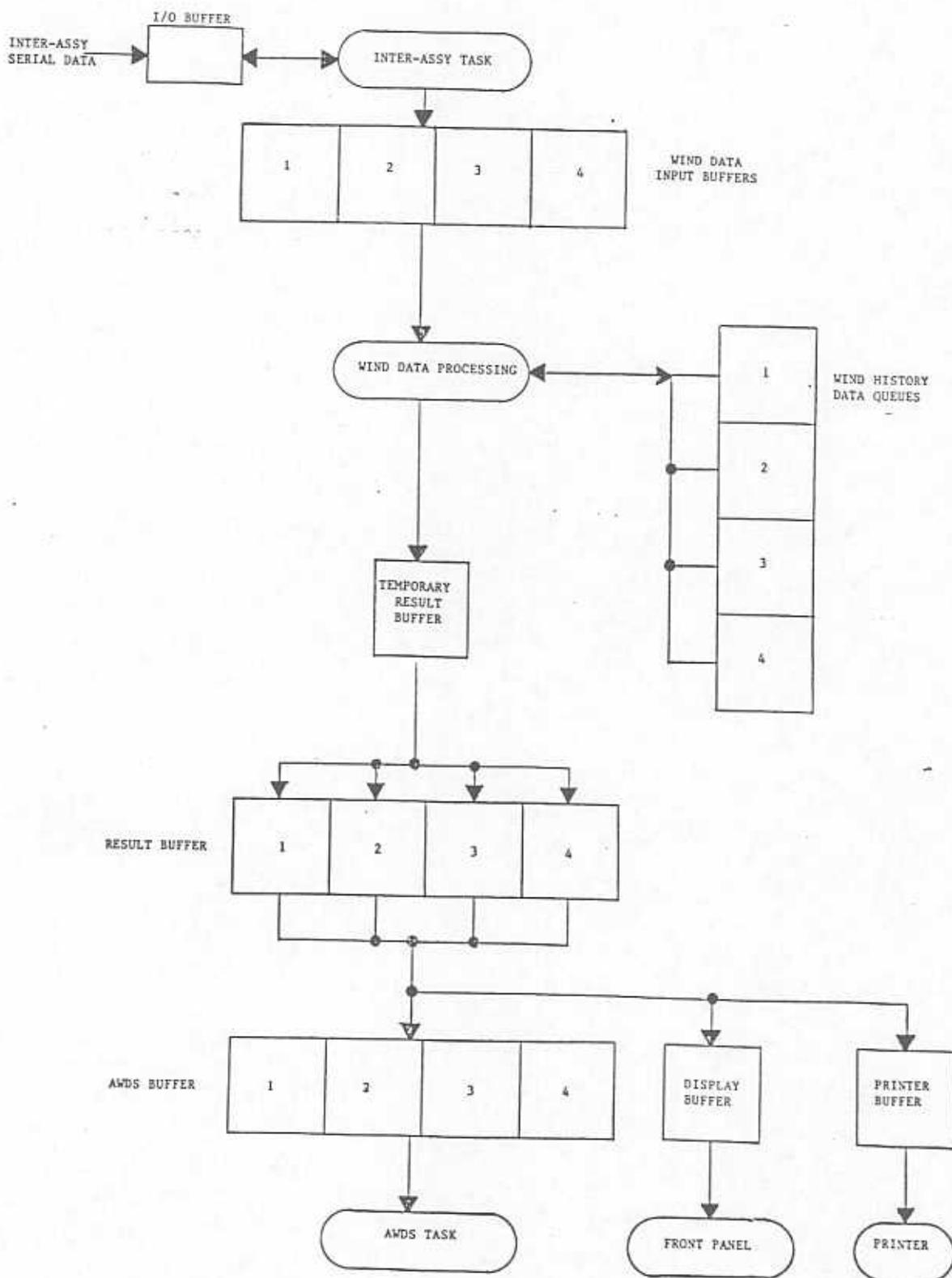


Figure 2  
CONTROL FLOW DIAGRAM

Figure 3

DATA FLOW DIAGRAM



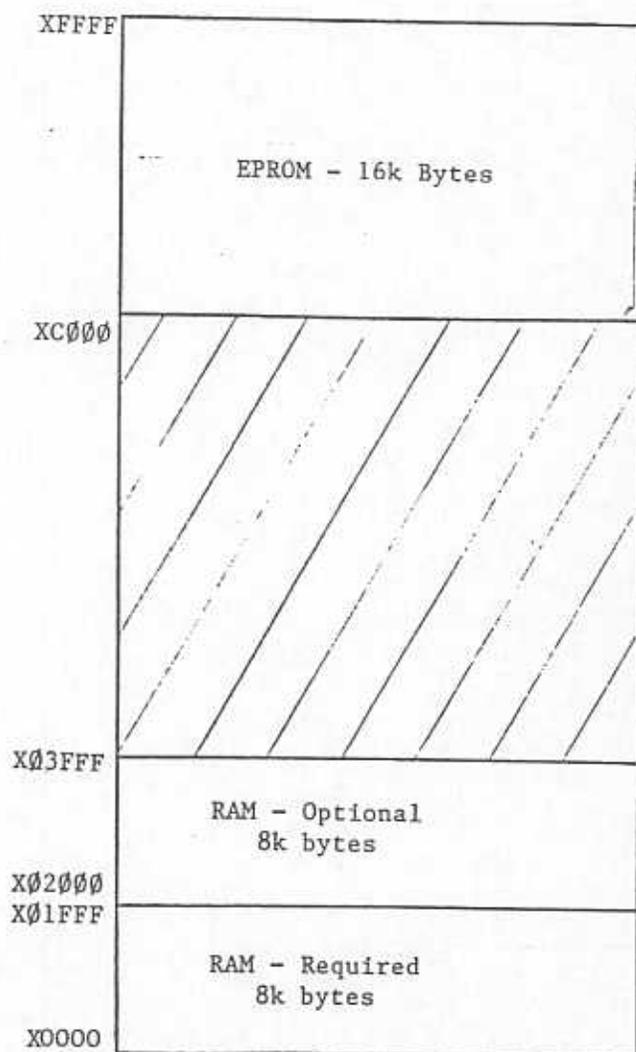


Figure 4  
MEMORY MAP

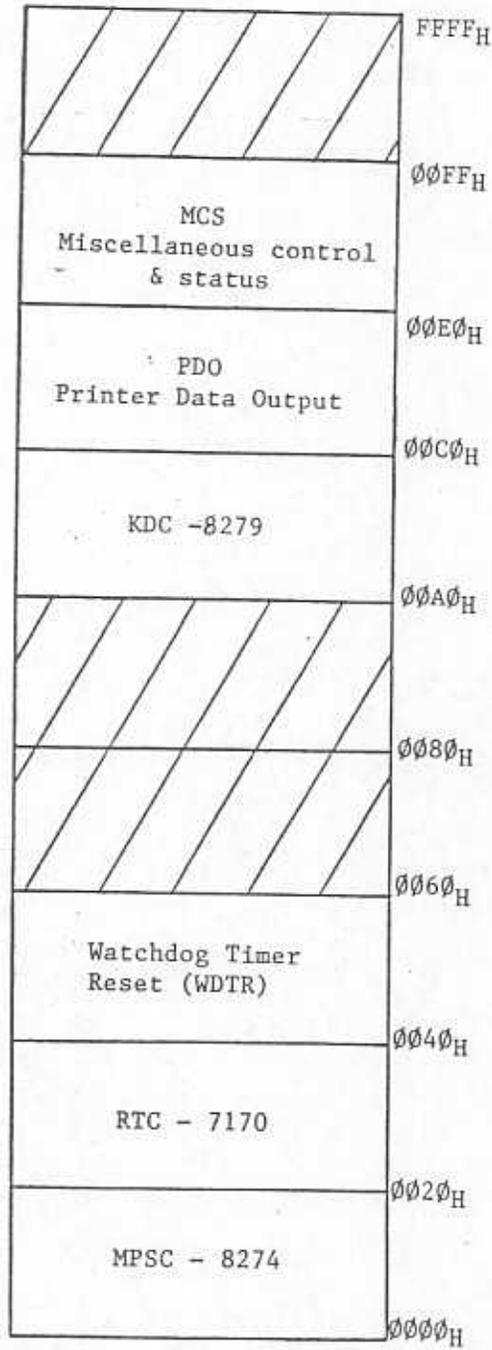
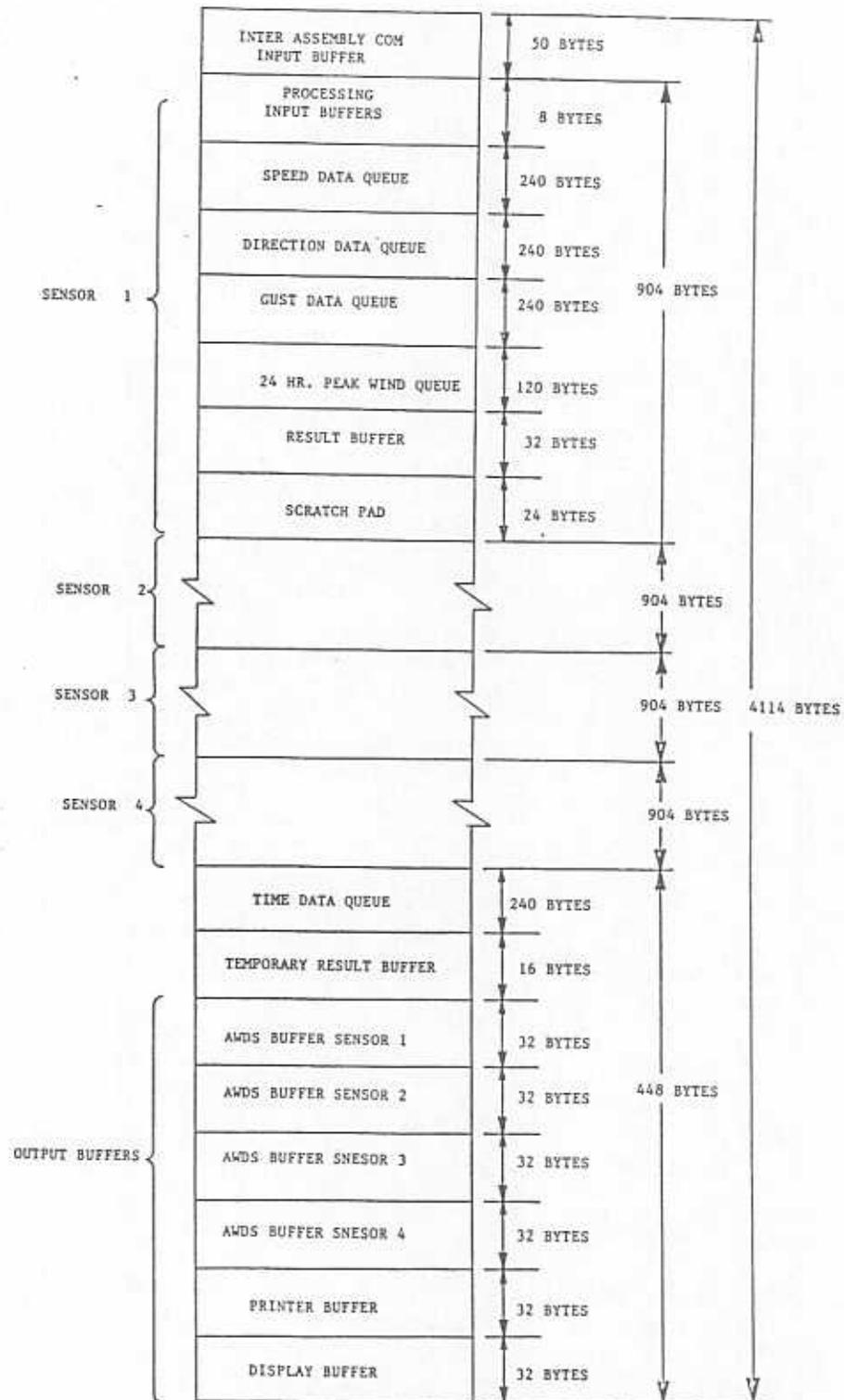


Figure 5

I/O MAP

Figure 6  
MEMORY ALLOCATION



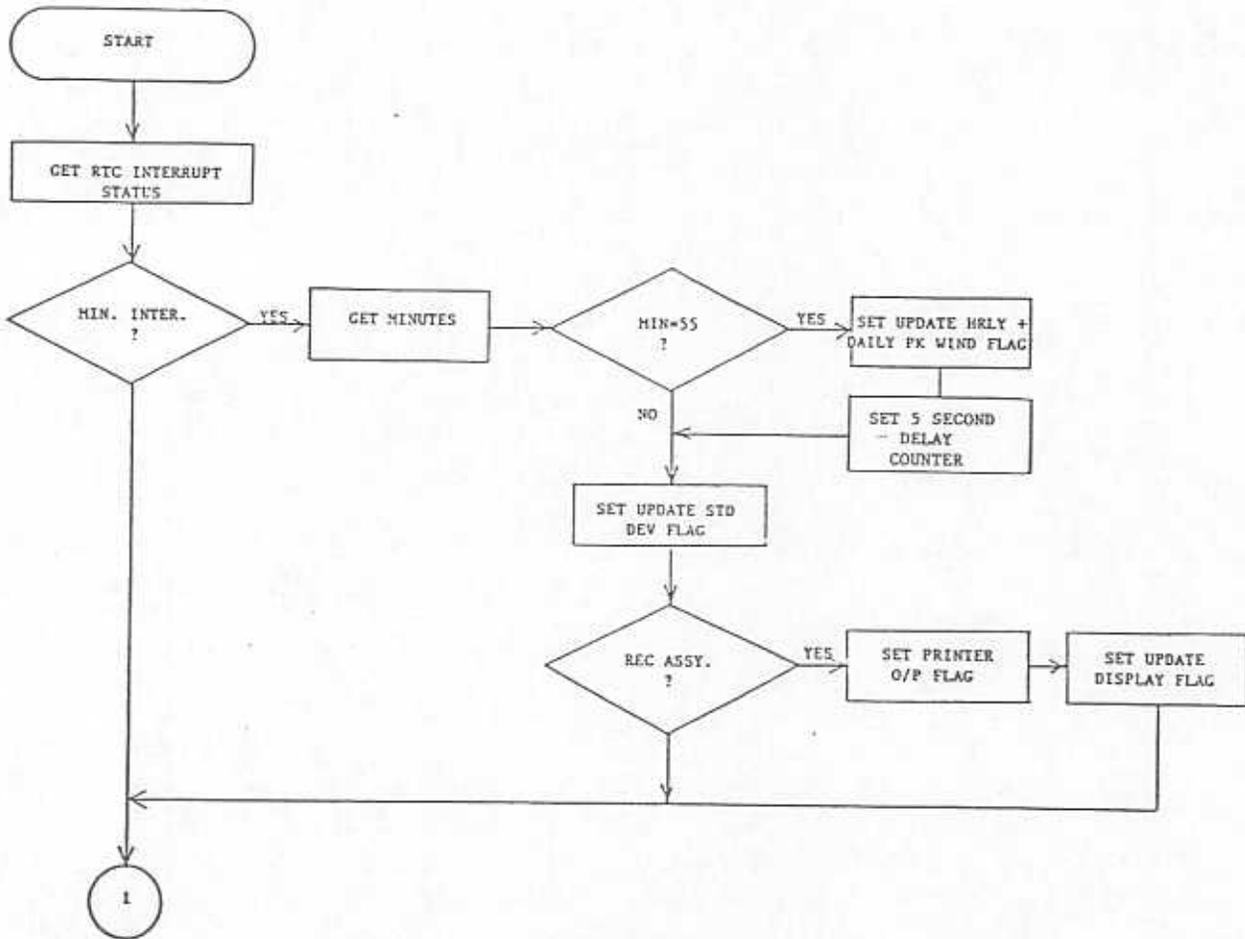


Figure 7  
SCHEDULER - Part I

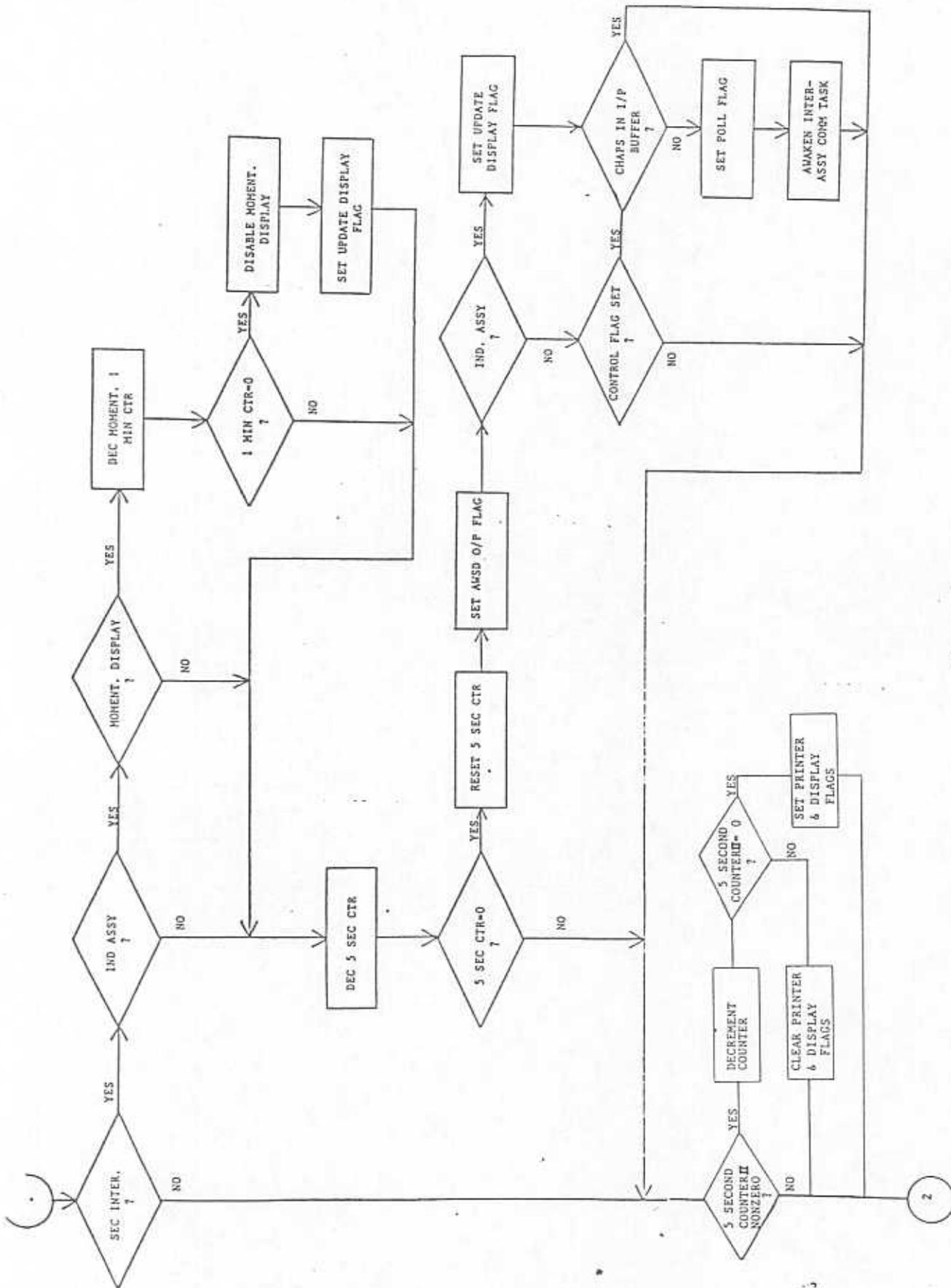


Figure 8  
SCHEDULER - PART 2

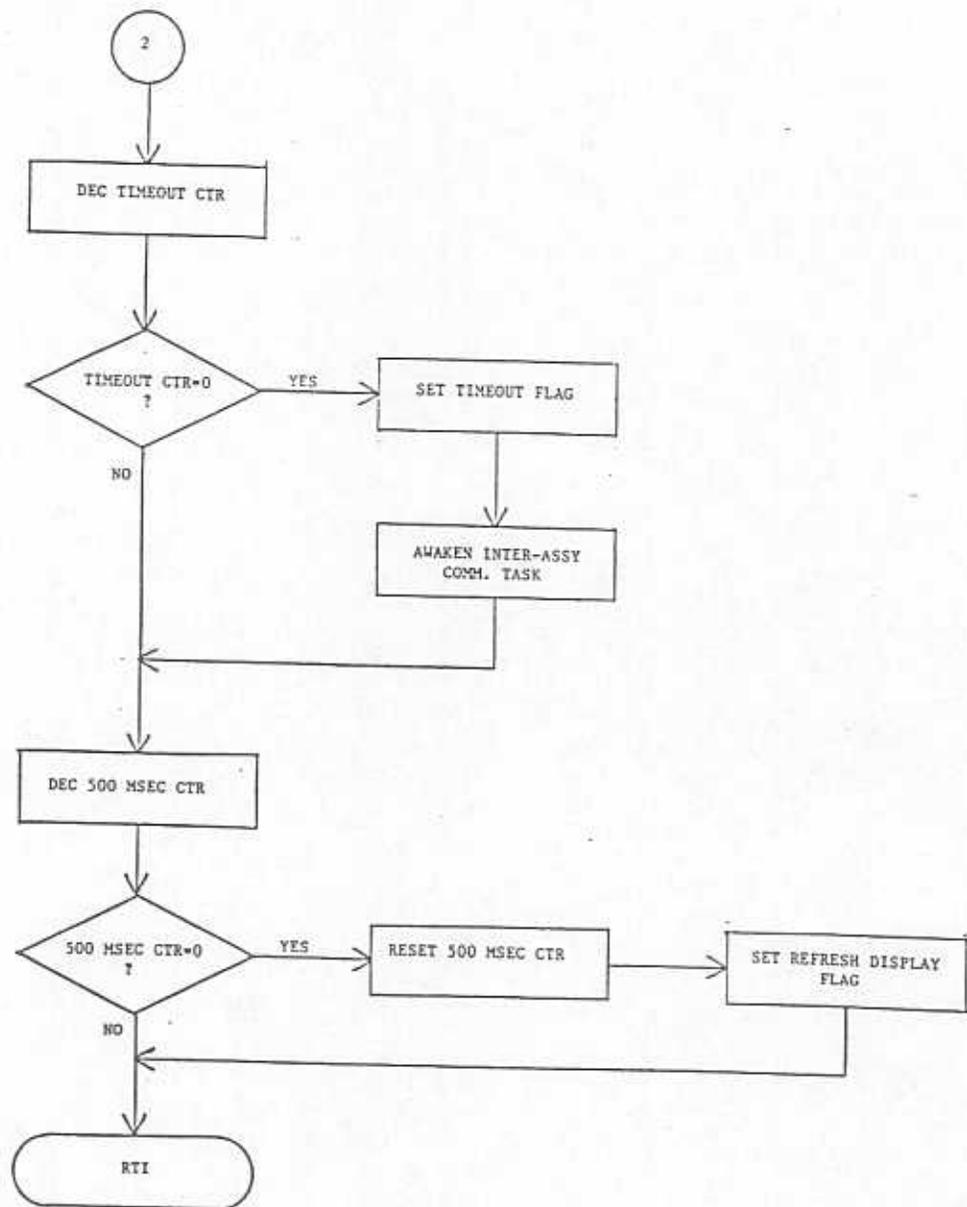


Figure 9

SCHEDULER - PART 3

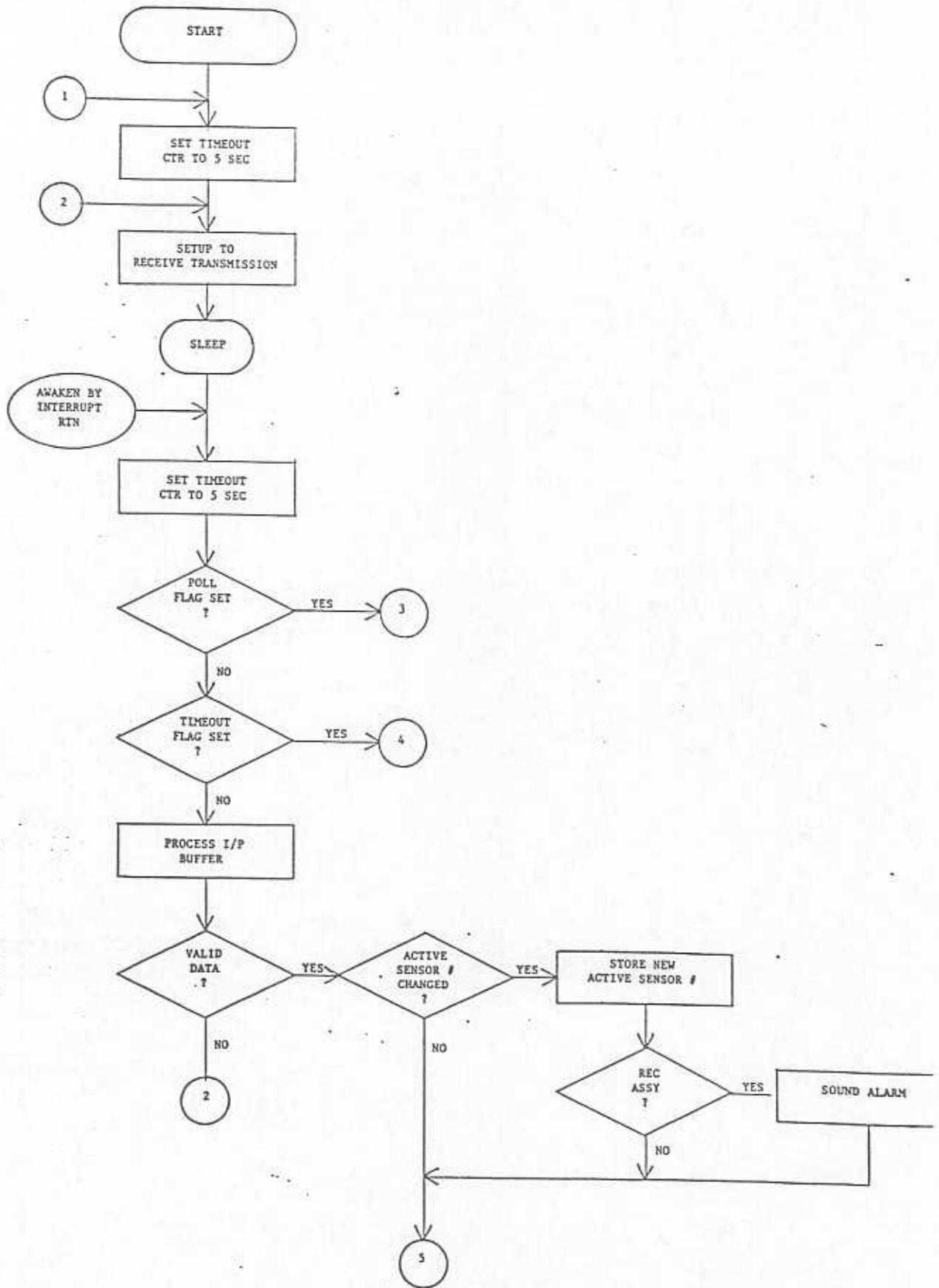
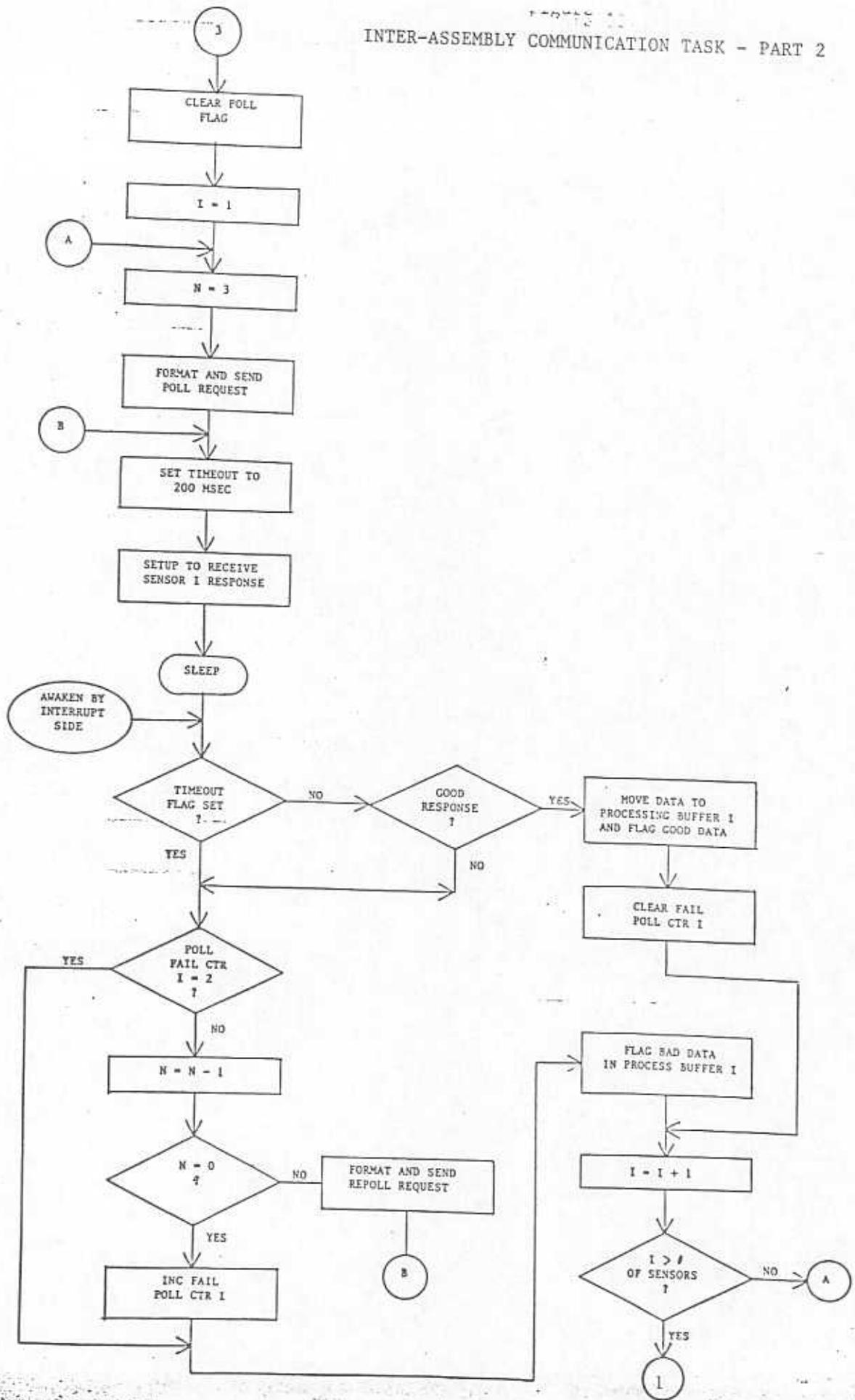


Figure 10

INTER-ASSEMBLY COMMUNICATION TASK - PART 2



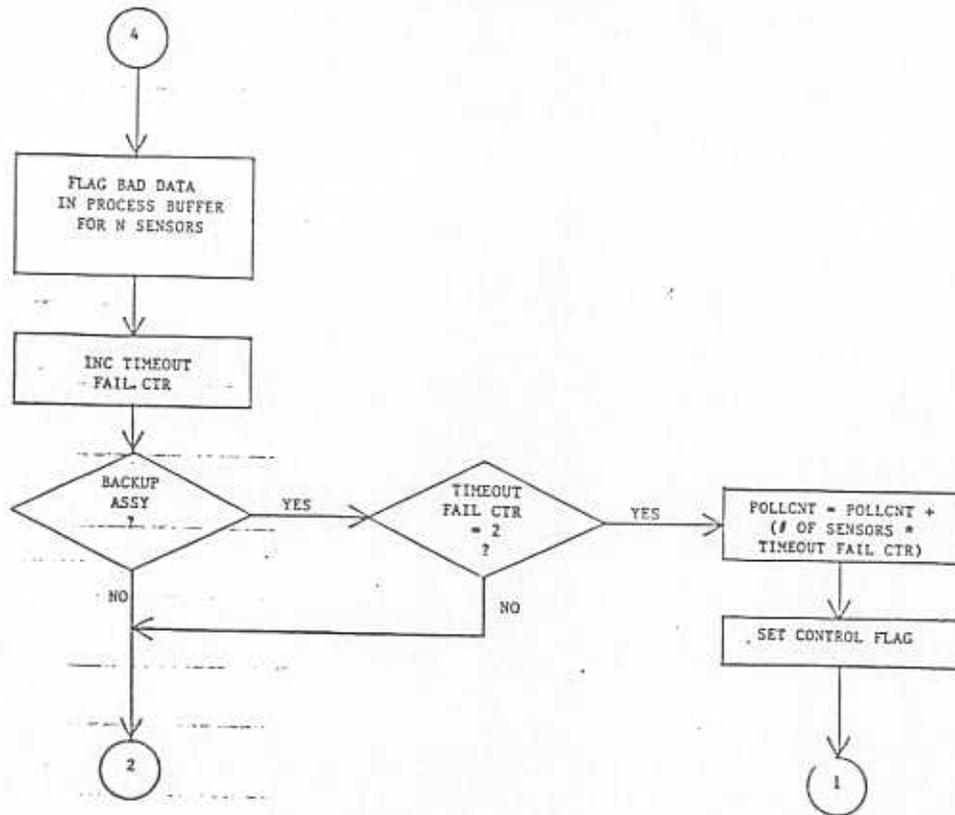


Figure 12

INTER-ASSEMBLY COMMUNICATION TASK - PART 3

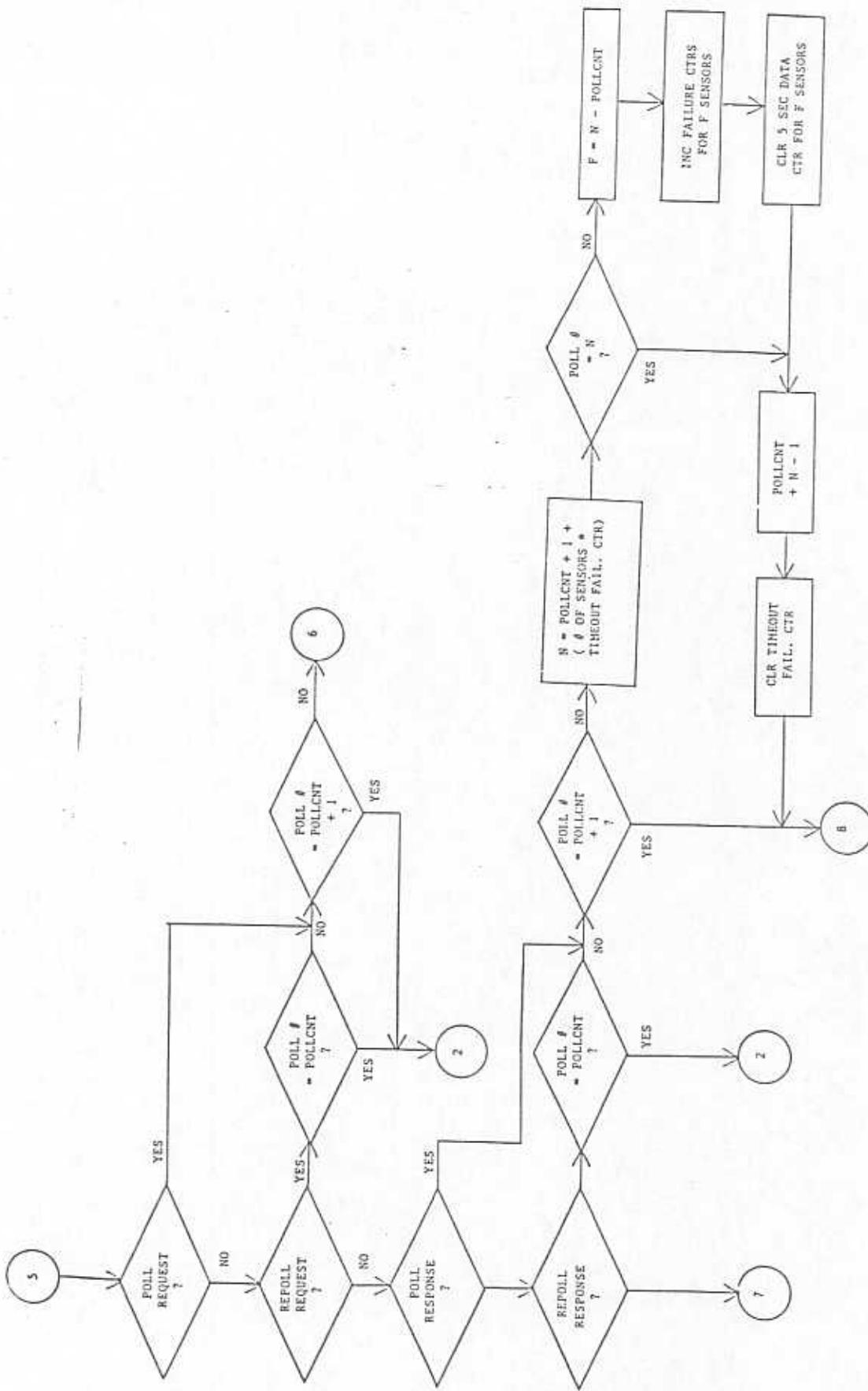
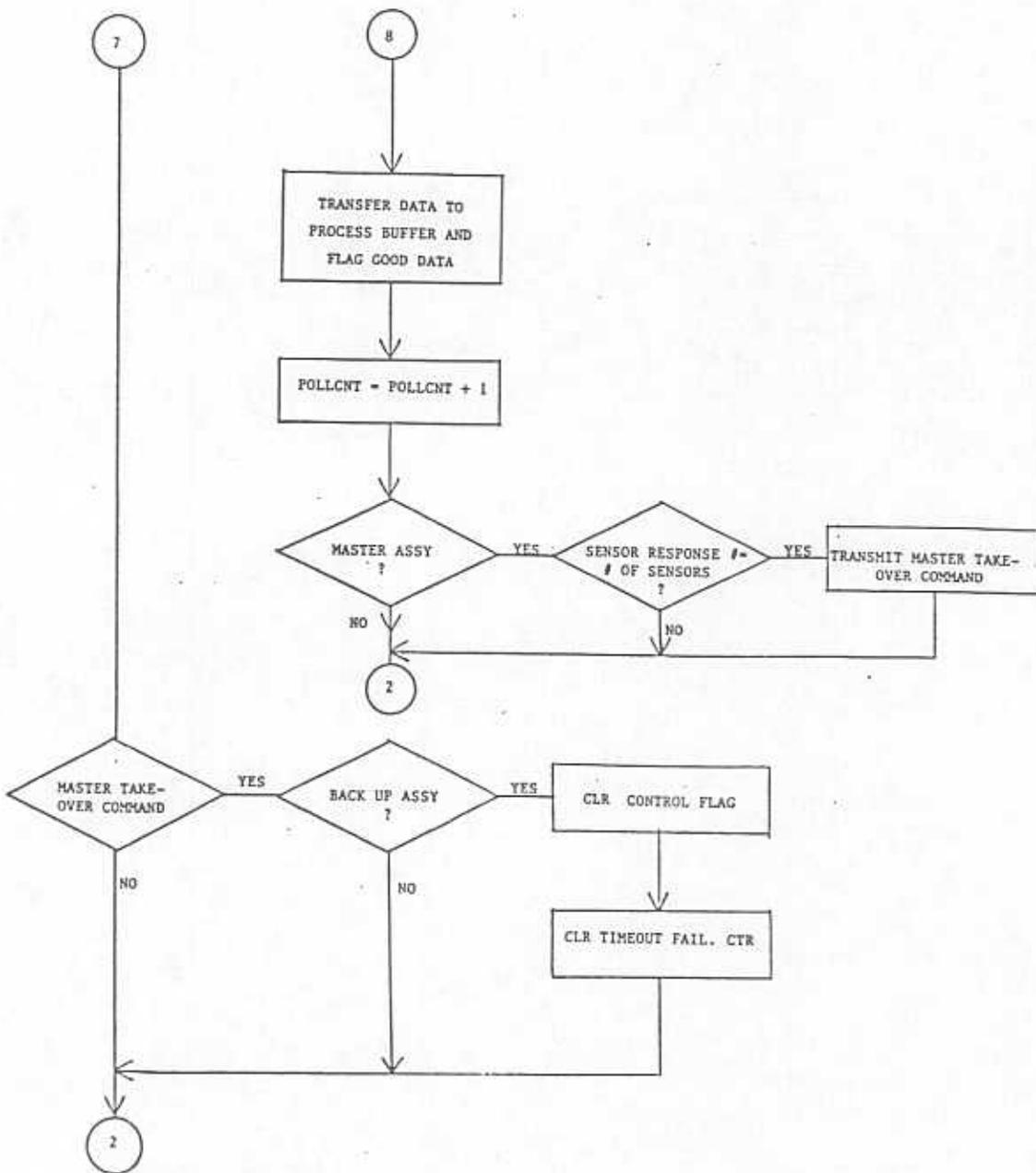


Figure 13

Figure 14

INTER-ASSEMBLY COMMUNICATION TASK - PART 5



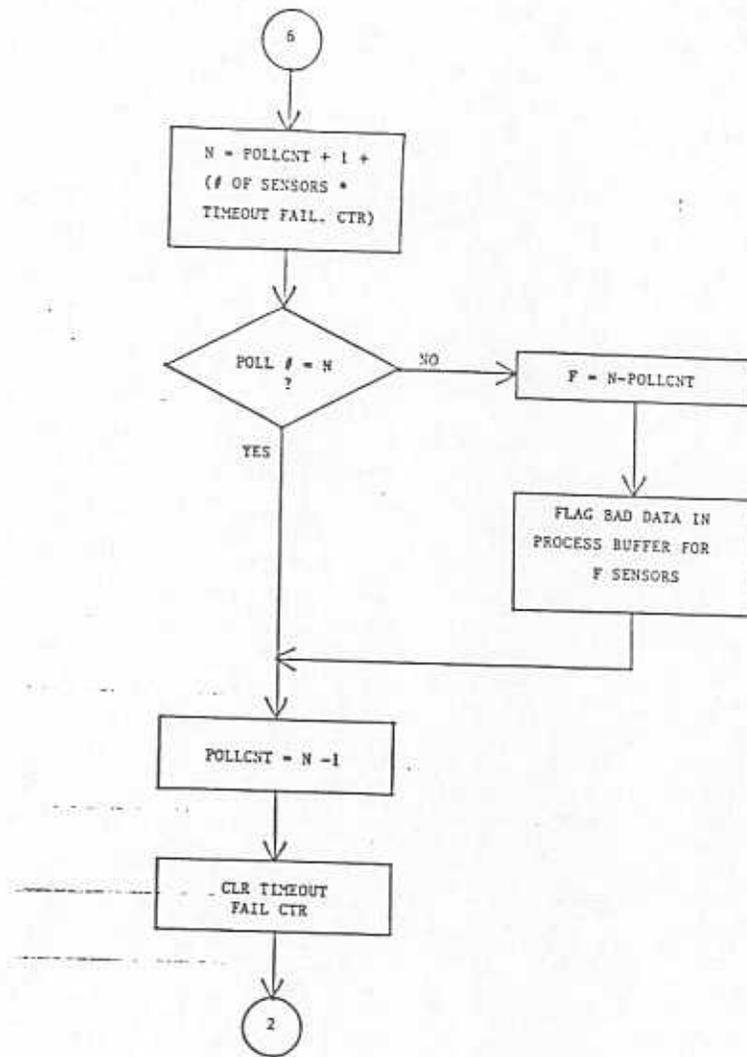


Figure 15

INTER-ASSEMBLY COMMUNICATION TASK - PART 6

FIGURE 16

INTER-ASSEMBLY COMMUNICATION INTERRUPT HANDLER

PART 1

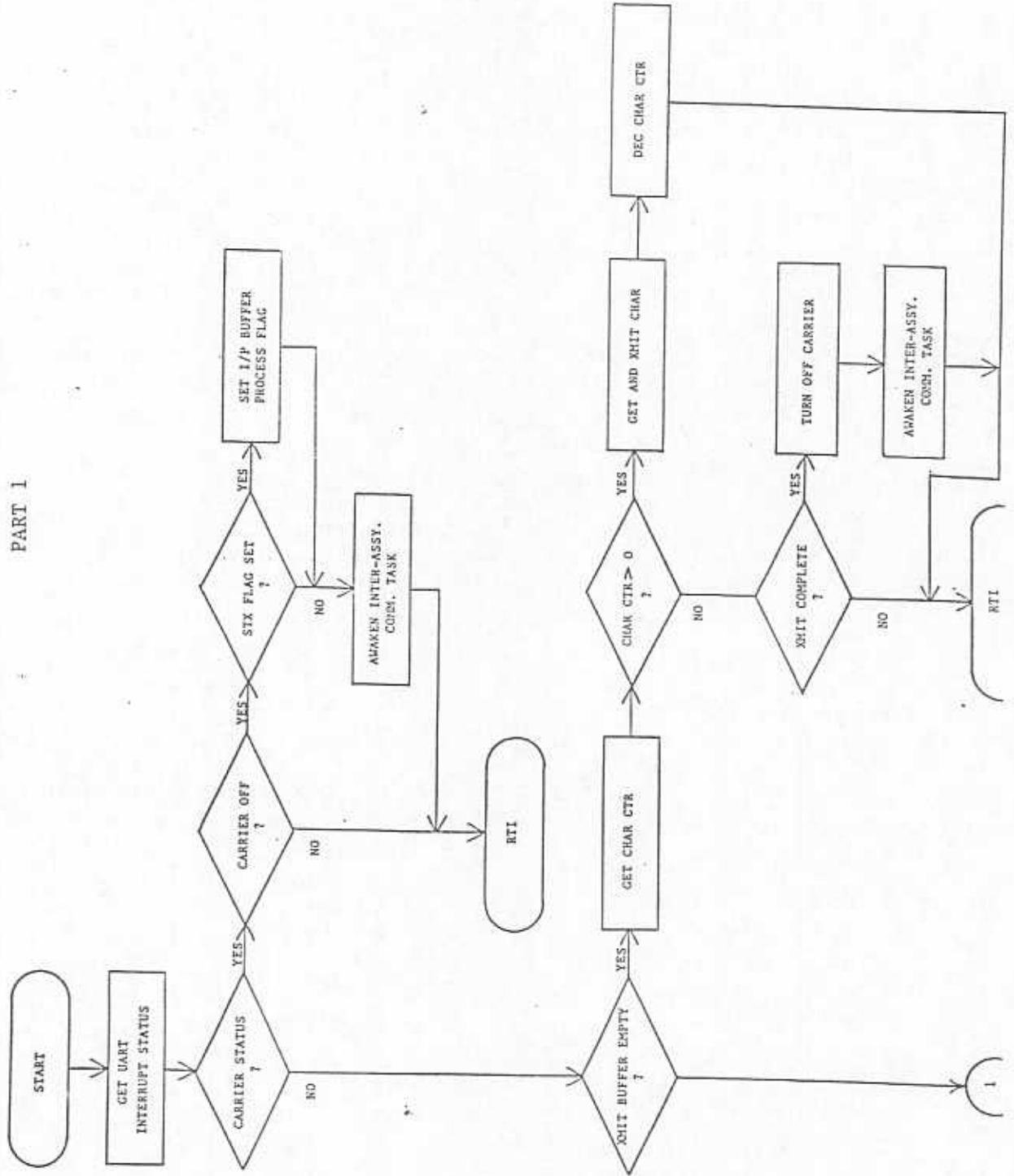


FIGURE 17

INTER-ASSEMBLY COMMUNICATION INTERRUPT HANDLER

PART 2

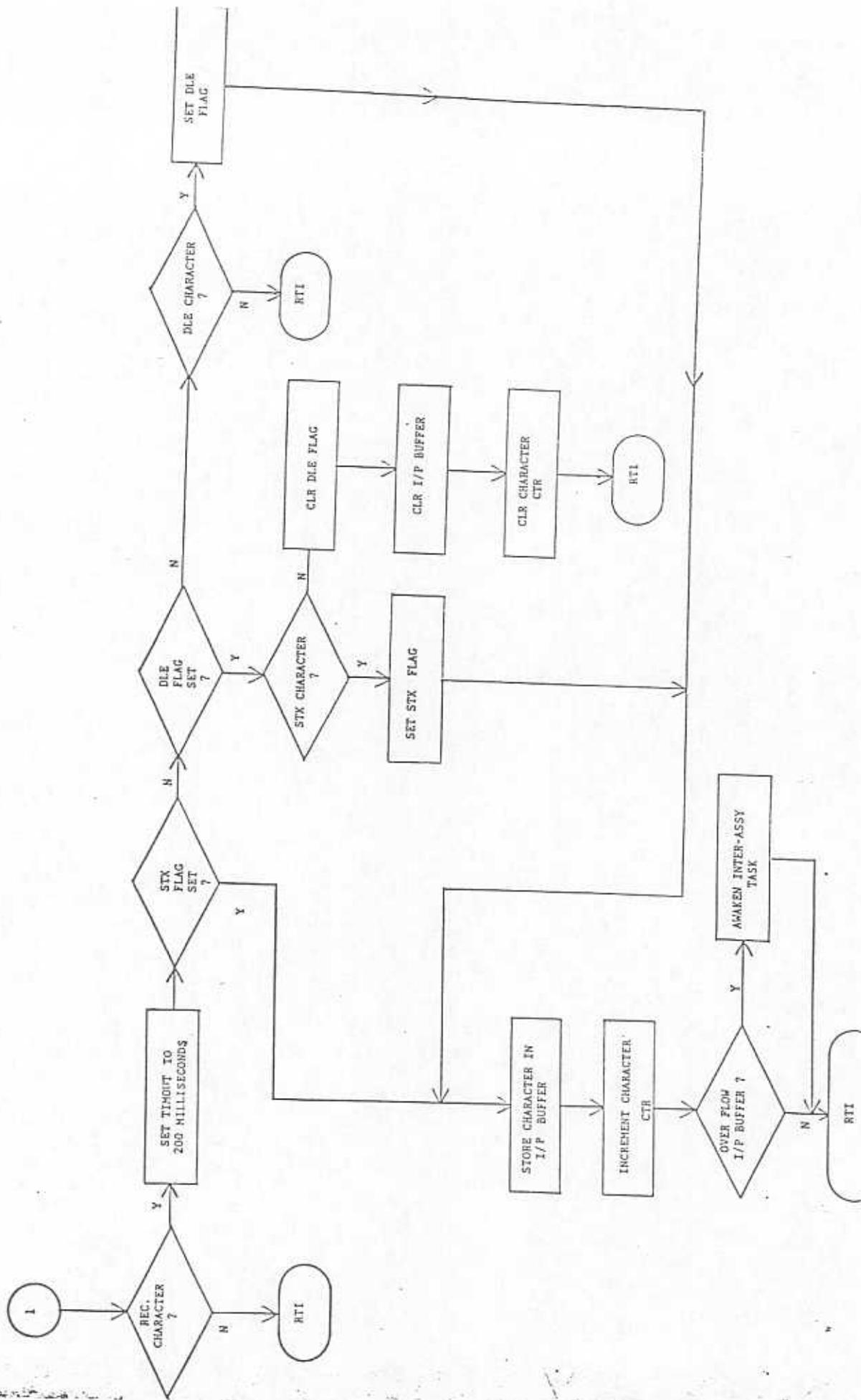


FIGURE 18

WIND DATA PROCESSING-MAJOR LOOP

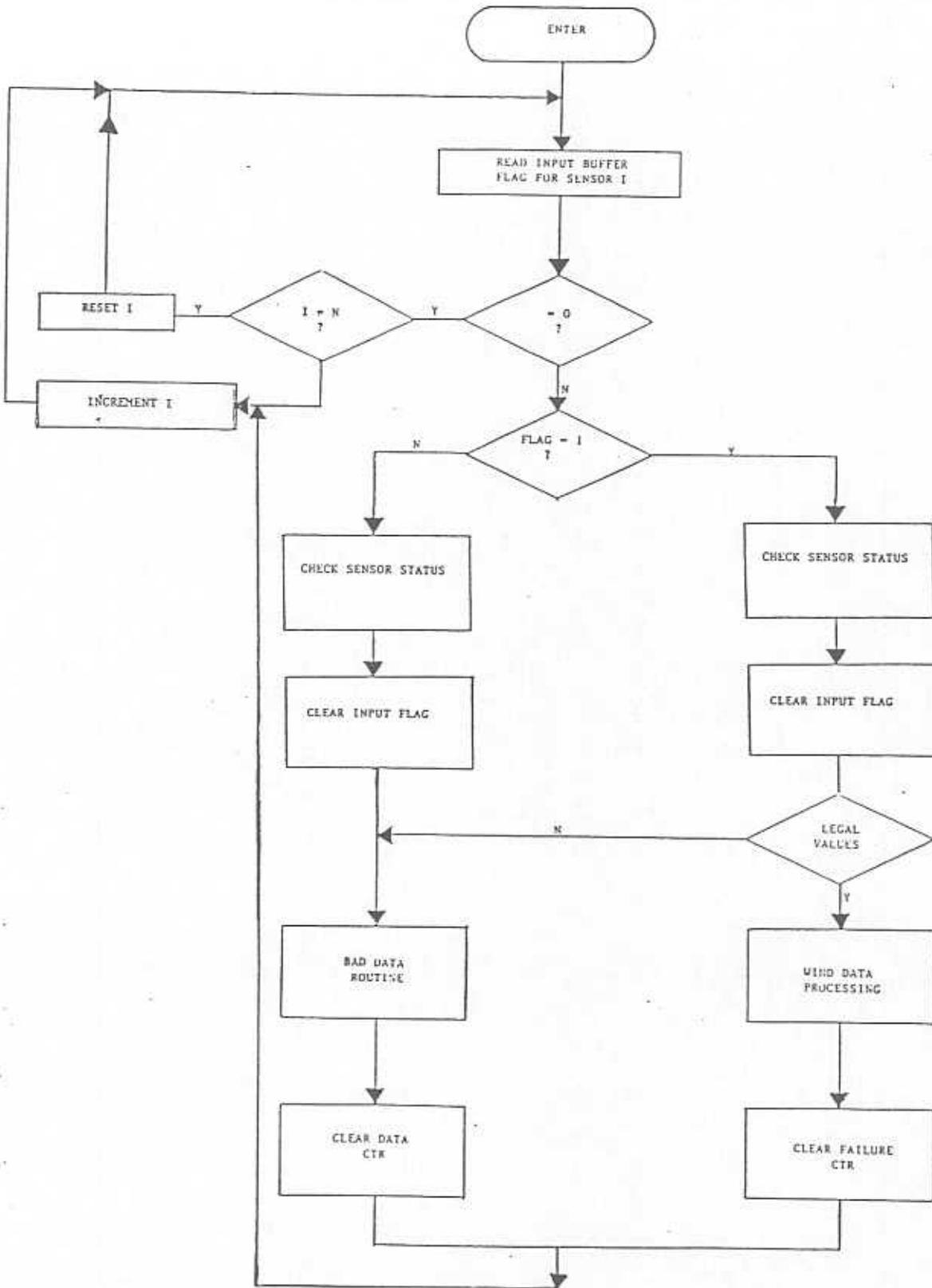


FIGURE 19

WIND DATA PROCESSING

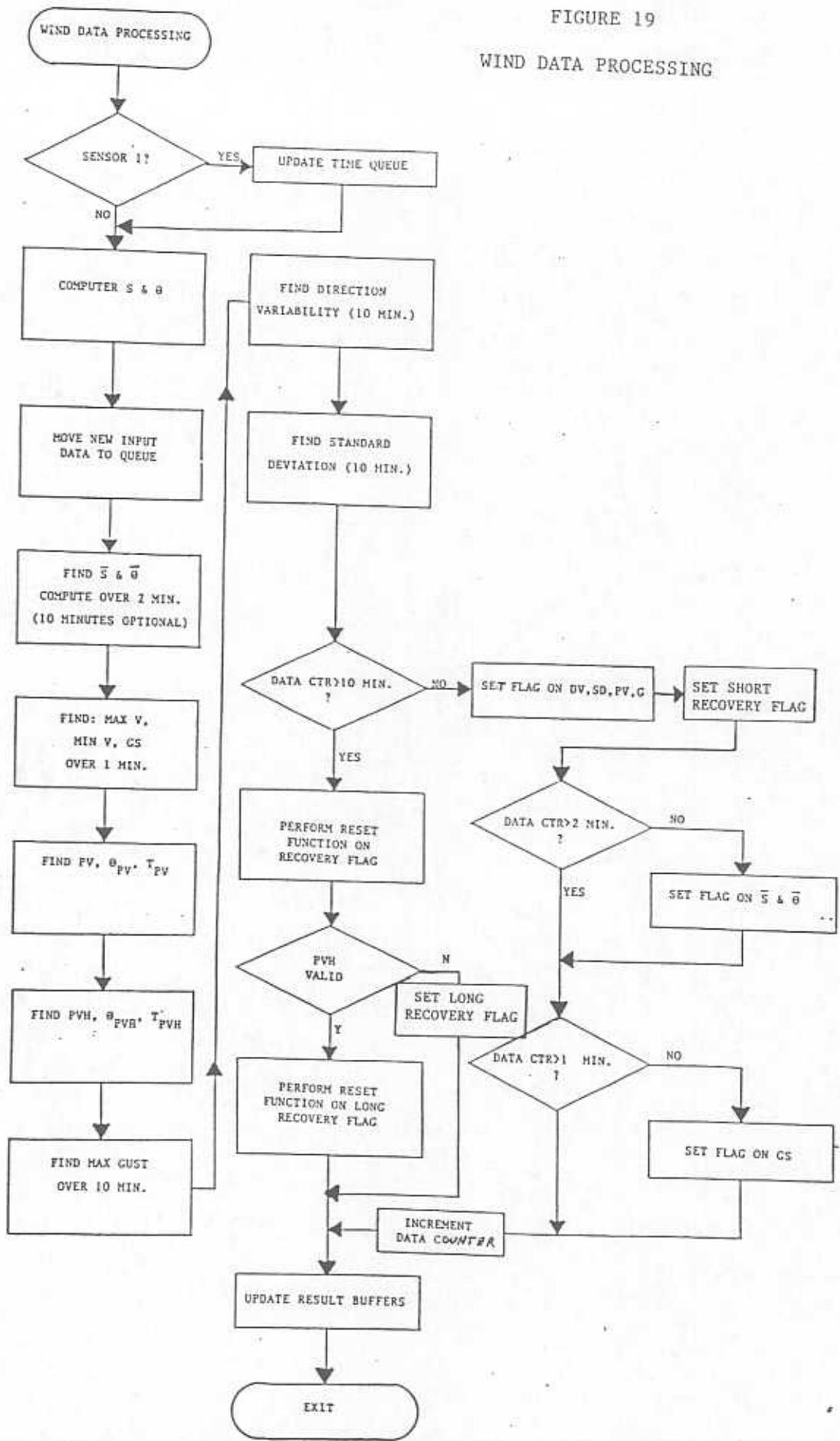


FIGURE 20  
FRAME OF REFERENCE

X + Y COMPONENTS OF WIND VELOCITY

VECTOR VALUES

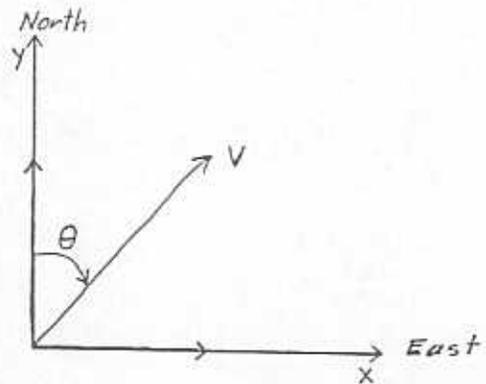
V = WIND SPEED

$\theta$  = WIND DIRECTION

ORTHOGONAL VALUES

X = V SIN ( $\theta$ )

Y = V COS ( $\theta$ )



DIRECTION ( $\theta$ ) IS REFERENCED TO THE NORTH AXIS AND INDICATES THE DIRECTION FROM WHICH THE WIND IS COMING.

FIGURE 21

## VECTORIAL AVERAGE ROUTINE

WIND SPEED & DIRECTION -  $\bar{S}, \bar{\theta}$ 

- 1) TWO MINUTE VECTORIAL AVERAGE OF FIVE SECOND SAMPLES.
- 2) UPDATE EVERY 5 SECONDS
- 3)  $0^{\circ}$  OR  $360^{\circ} = 360^{\circ}$
- 4) CALM WIND = 0 DEGREES & 0 SPEED

QUEUE SEGMENT

X-COMPONENT DATA -  $X_1 X_2 X_3 \dots X_{24}$ Y-COMPONENT DATA -  $Y_1 Y_2 Y_3 \dots Y_{24}$ 

EQUATIONS

2 AND 10 MINUTE AVERAGE VALUES

$$\bar{X}_2 = \sum_{i=1}^{24} X_i / 24 \quad \bar{X}_{10} = \sum_{i=1}^{120} X_i / 120$$

$$\bar{Y}_2 = \sum_{i=1}^{24} Y_i / 24 \quad \bar{Y}_{10} = \sum_{i=1}^{120} Y_i / 120$$

2 AND 10 MINUTE VECTORIAL AVERAGE

$$\bar{S}_2 = \sqrt{\bar{X}_2^2 + \bar{Y}_2^2} \quad \bar{S}_{10} = \sqrt{\bar{X}_{10}^2 + \bar{Y}_{10}^2}$$

2 AND 10 MINUTE AVERAGE DIRECTION

$$\bar{\theta} = \text{ARCTAN} \frac{\bar{X}}{\bar{Y}}, \text{ for } +X, +Y$$

$$\bar{\theta} = \text{ARCTAN} \frac{|\bar{Y}|}{|\bar{X}|} + 90^{\circ}, \text{ for } +X, -Y$$

$$\bar{\theta} = \text{ARCTAN} \frac{|\bar{X}|}{|\bar{Y}|} + 180^{\circ}, \text{ for } -X, -Y$$

$$\bar{\theta} = \text{ARCTAN} \frac{|\bar{Y}|}{|\bar{X}|} + 270^{\circ}, \text{ for } -X, +Y$$

FIGURE 22

CALCULATION OF GUST SPREAD

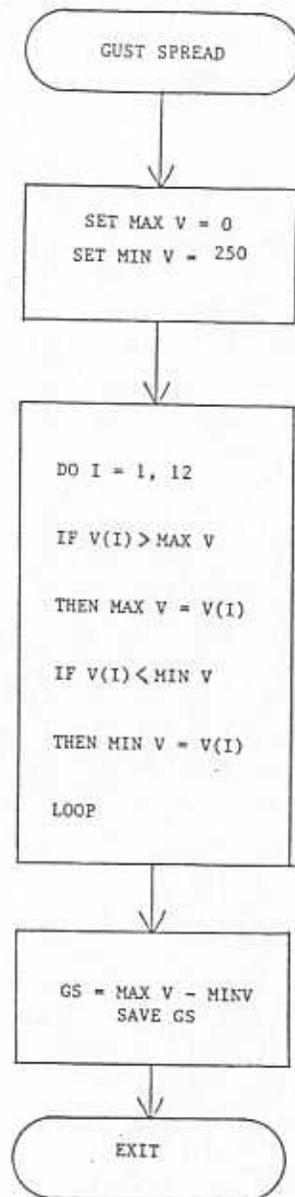
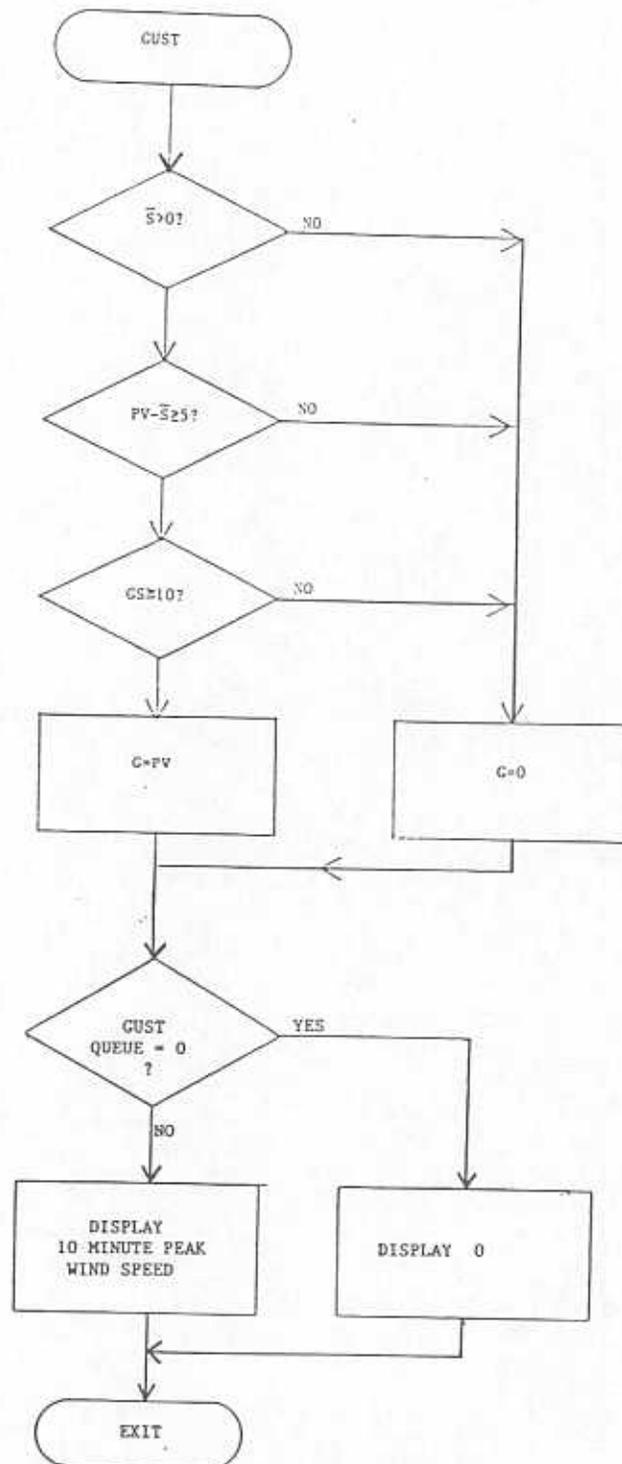


FIGURE 23  
CALCULATION OF GUST  
USING TWO MINUTE AVERAGES



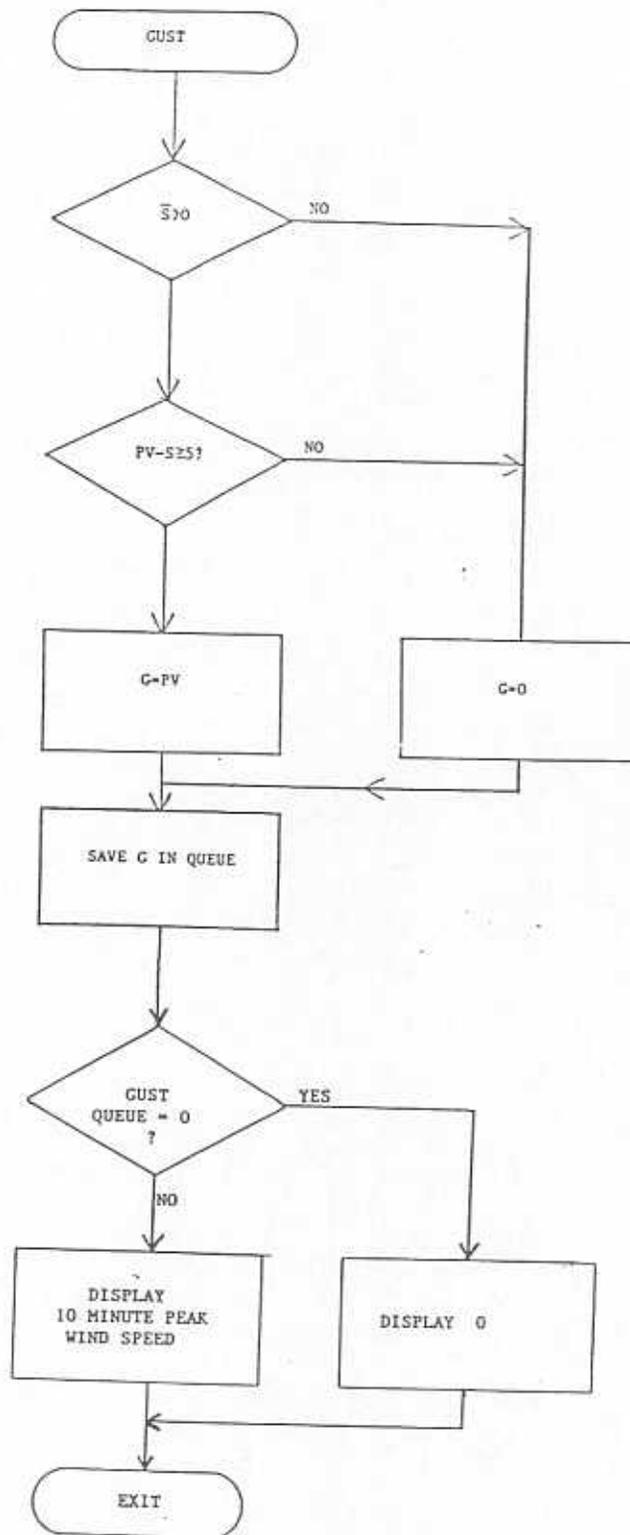
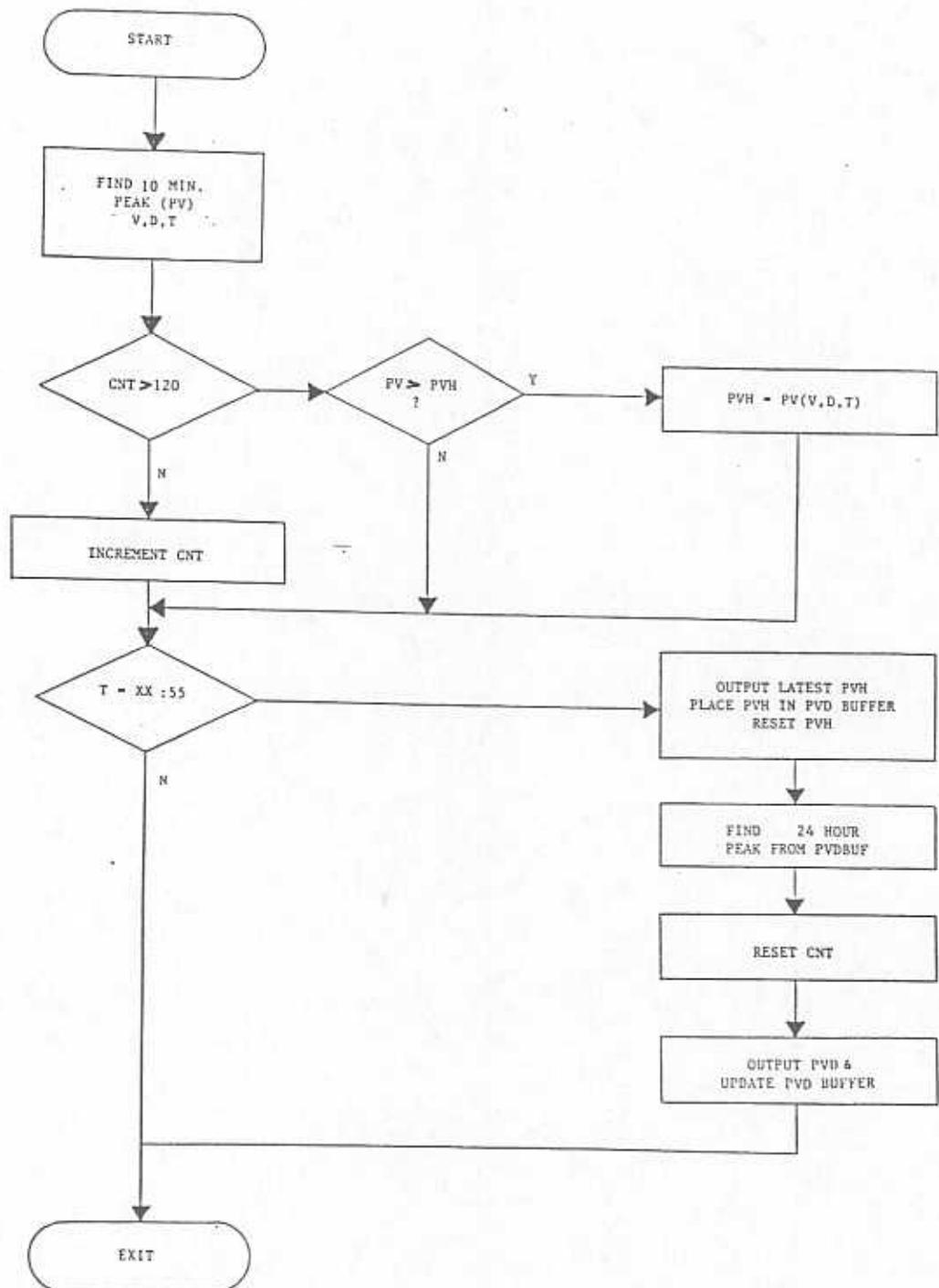


FIGURE 24  
 CALCULATION OF GUST  
 USING TEN MINUTE AVERAGES

FIGURE 25  
PEAK WIND CALCULATIONS



DIRECTION VARIABILITY - DV

COMPUTE  $D\theta = \theta_n - \theta_0$

IF  $D\theta > \pm 180^\circ$ ; ADD  $\mp 360^\circ$

COMPUTE  $DD\theta = D\theta_i - D\theta_{i-1}$

IF  $DD\theta > \pm 180^\circ$ ; ADD  $+ 360^\circ$  TO  $D\theta$

IF  $D\theta$  IS NEW MAX OR MIN, SAVE

IF  $\text{MAX} - \text{MIN} \geq 360^\circ$ , THEN  $DV1 = DV2 = \bar{\theta}$

ELSE  $DV1 = \theta_0 + \text{MIN}$

$DV1 = \theta_0 + \text{MIN}$

IF  $DV1 < 0$  ADD  $360^\circ$

$DV2 = \theta_0 + \text{MAX}$

IF  $DV2 > 360^\circ$  SUBTRACT  $360^\circ$

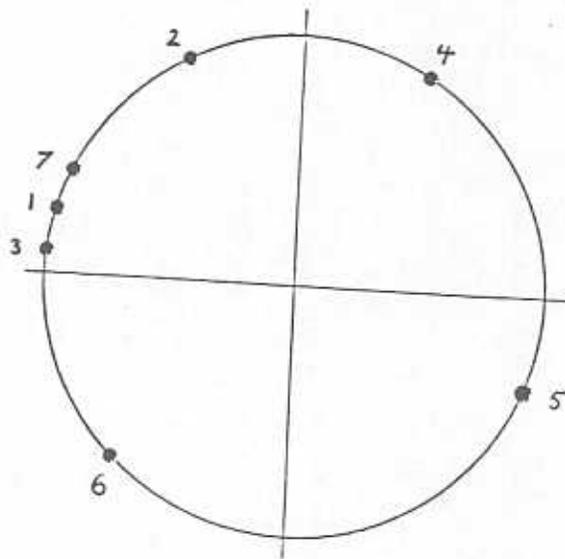
Figure 26

DIRECTION VARIABILITY

Figure 27(A)

DIRECTIONAL VARIABILITY CALCULATION - EXAMPLE 1

	$\theta$	$D\theta$	$D\theta_i - D\theta_{i-1}$	<u>CCW (min)</u>	<u>CW (max)</u>	<u>max-min</u>
1	285	0	0	0	0	--
2	350	65	65	0	65	--
3	275	-10	-75	-10	65	--
4	10	85 (-275)	75	-10	85	--
5	120	195 (-165)	(-250)	-10	195	--
6	210	285 (-75)	(-270)	-10	285	--
7	295	370 (10)	(-275)	-10	370	380



$$DV1 = DV2 = \bar{\theta}$$

( ) Intermediate Value

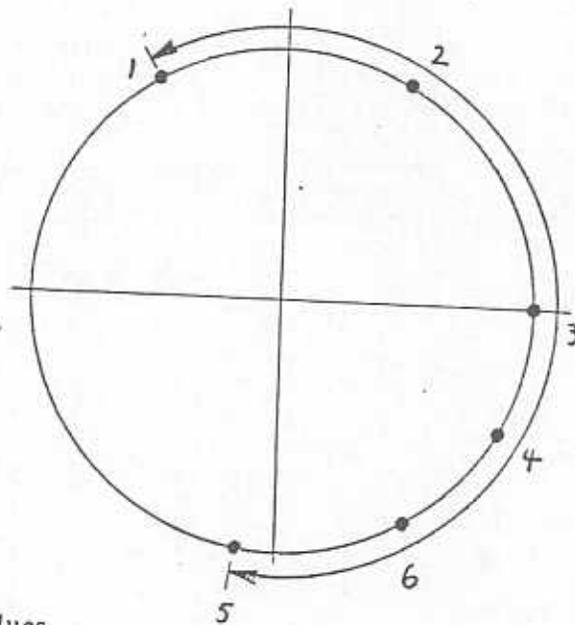
Figure 27(B)

DIRECTIONAL VARIABILITY CALCULATION - EXAMPLE 2

	$\theta$	$D\theta$	$D\theta_i - D\theta_{i-1}$	<u>CCW (min)</u>	<u>CW (max)</u>	<u>max-min</u>
1	340	0	0	0	0	--
2	20	40 (-320)	40	0	40	--
3	90	110 (-250)	70	0	70	--
4	120	140 (-220)	30	0	140	--
5	185	205 (-155)	(-295)	0	205	205
6	130	150 (-210)	-55	0	205	205

$$DVI = \theta_0 + \text{MIN} = 340$$

$$DV2 = \theta_0 + \text{MAX} = (545) - 360 = 185$$



( ) Intermediate Values

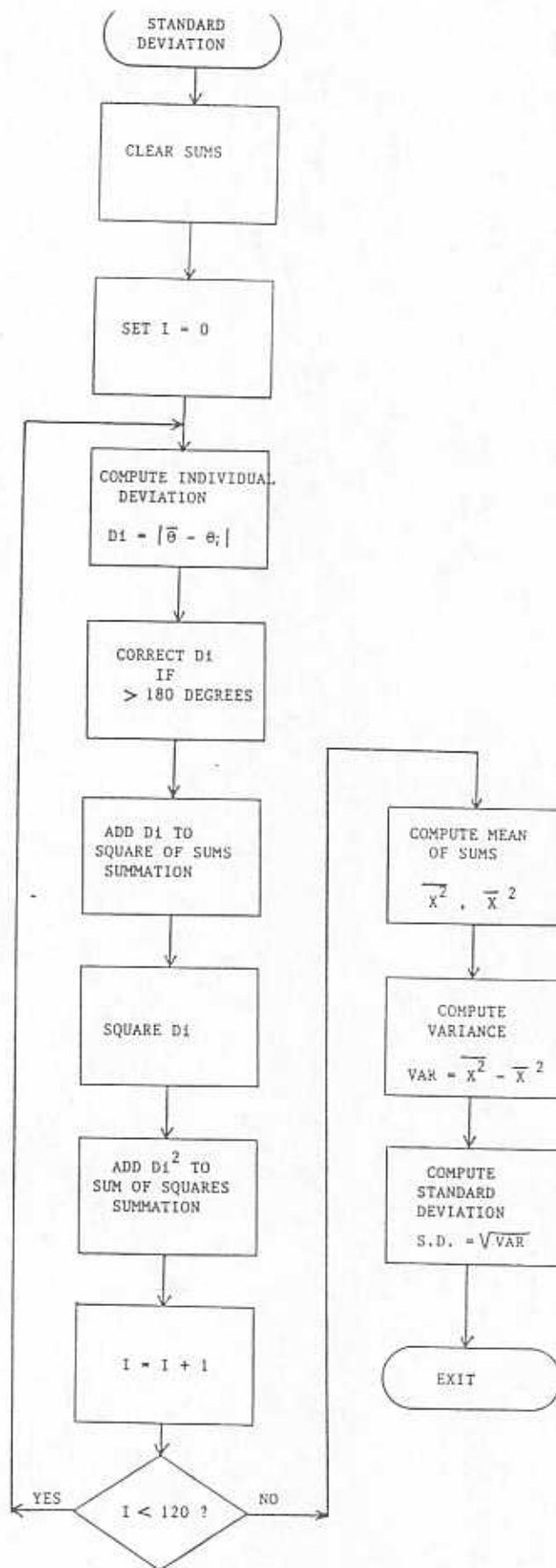


Figure 28 - Standard Deviation of Wind Direction Routine

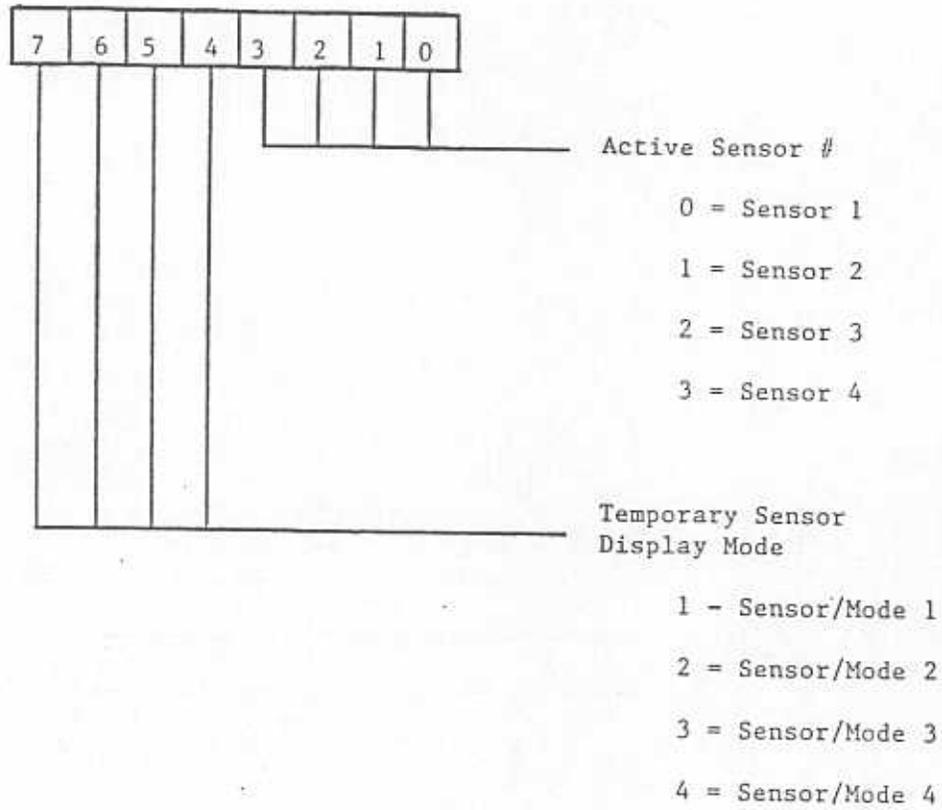


Figure 29

SENSOR # FLAG BIT MAP

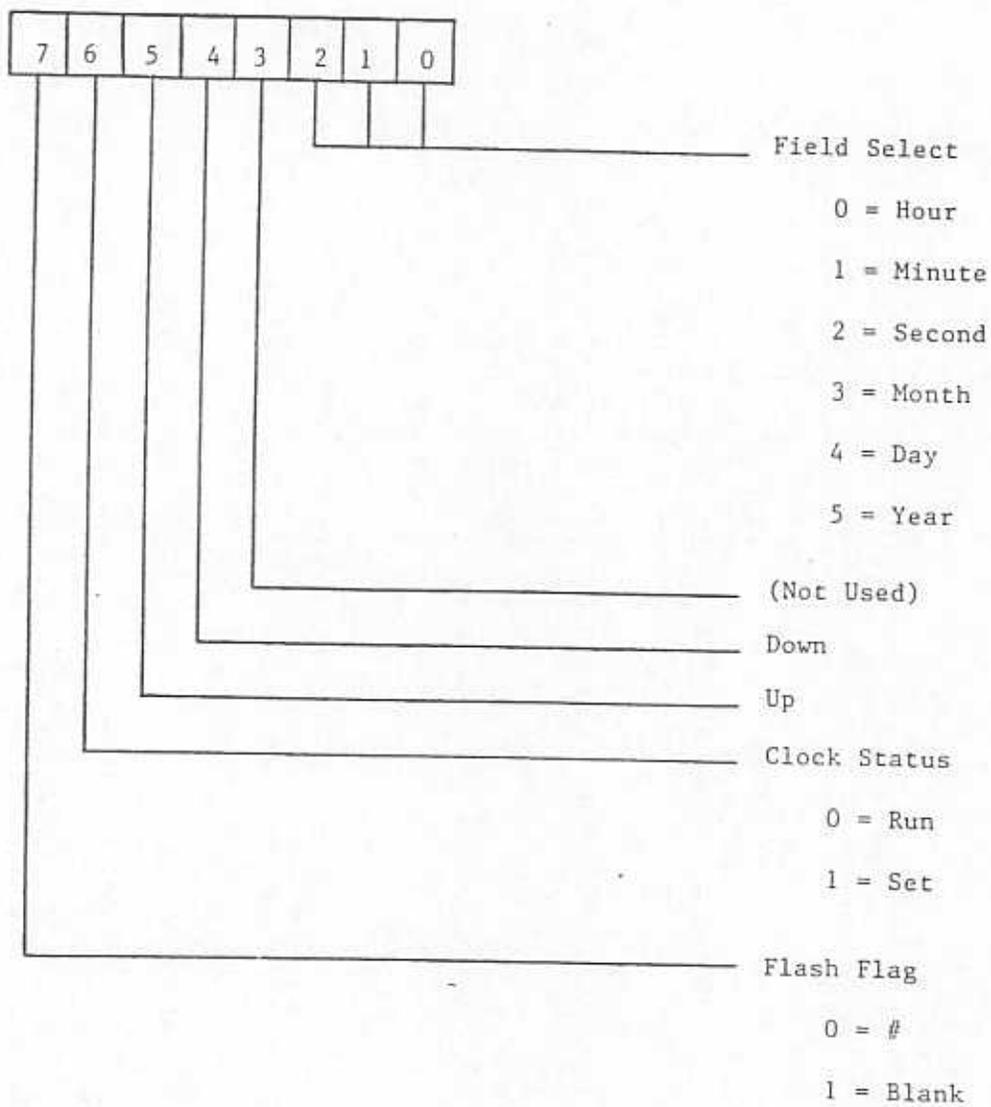


Figure 30  
 CLOCKFLAG BIT MAP

Status Bit

1	1	0
---	---	---

Acknowledge Bit

0	1	0
---	---	---

No Error

Flash Display

Error Acknowledged

Figure 31

ACKNOWLEDGE/STATUS BIT COMBINATION

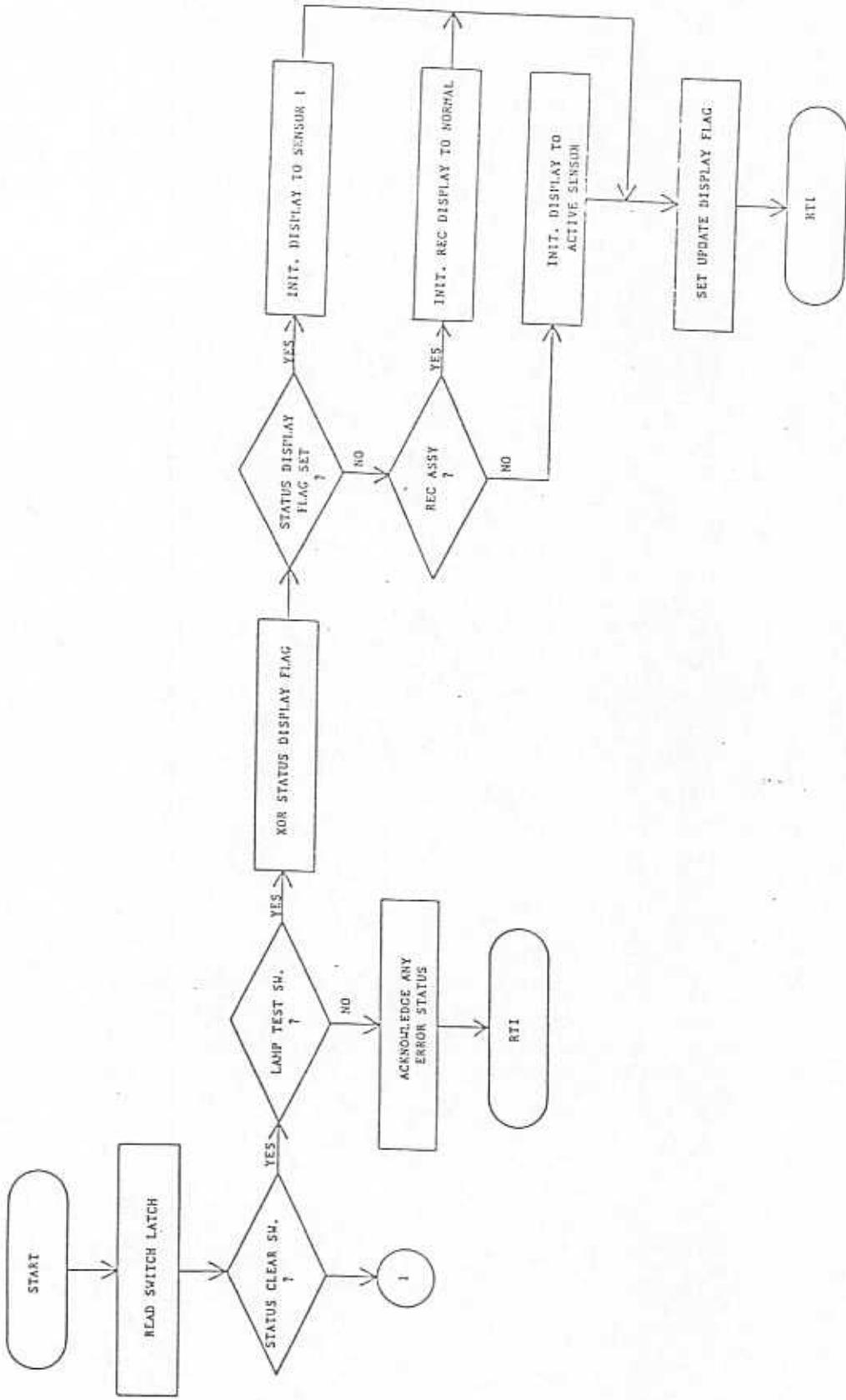
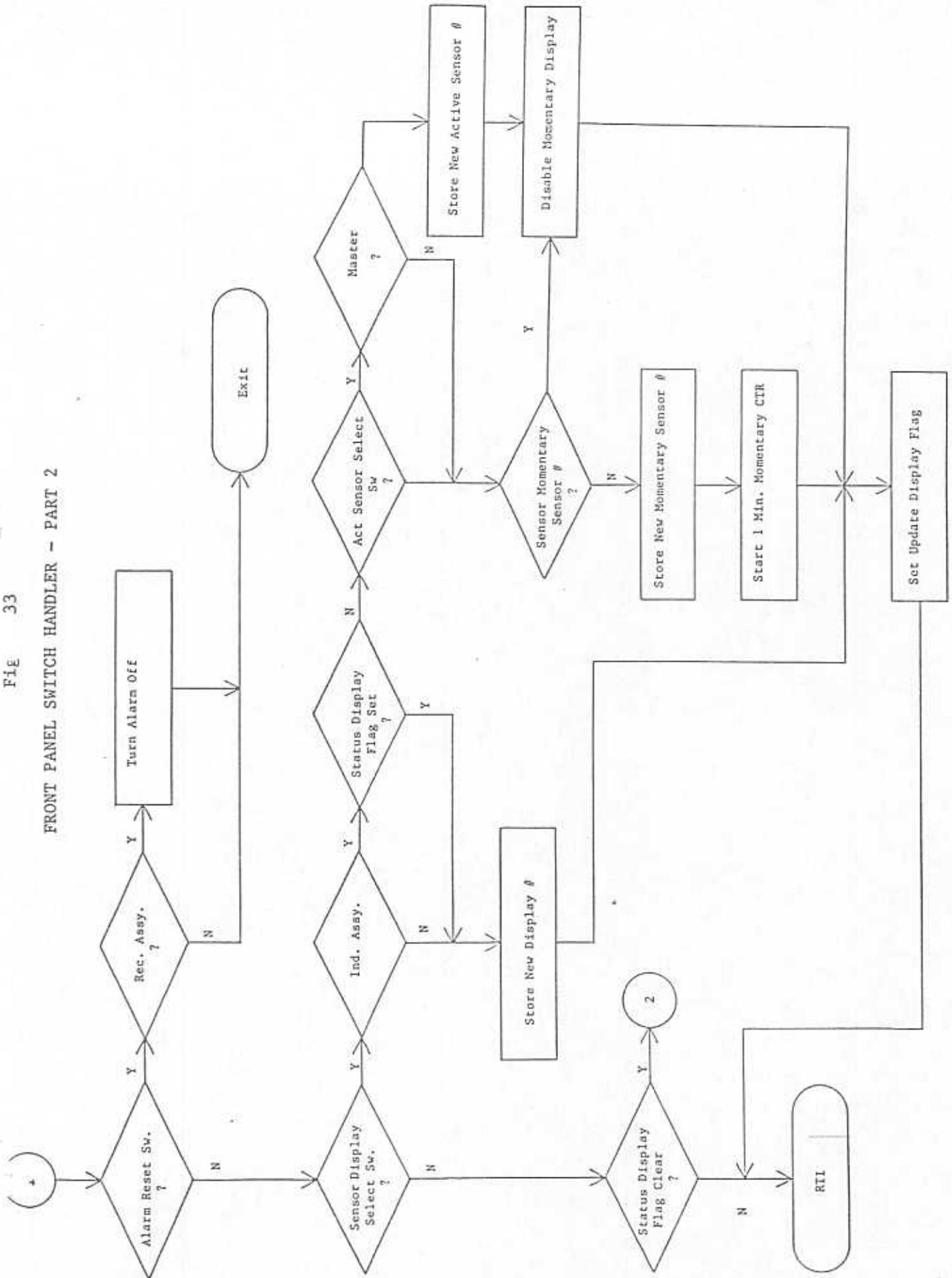


Figure 32  
FRONT PANEL SWITCH HANDLER - PART 1



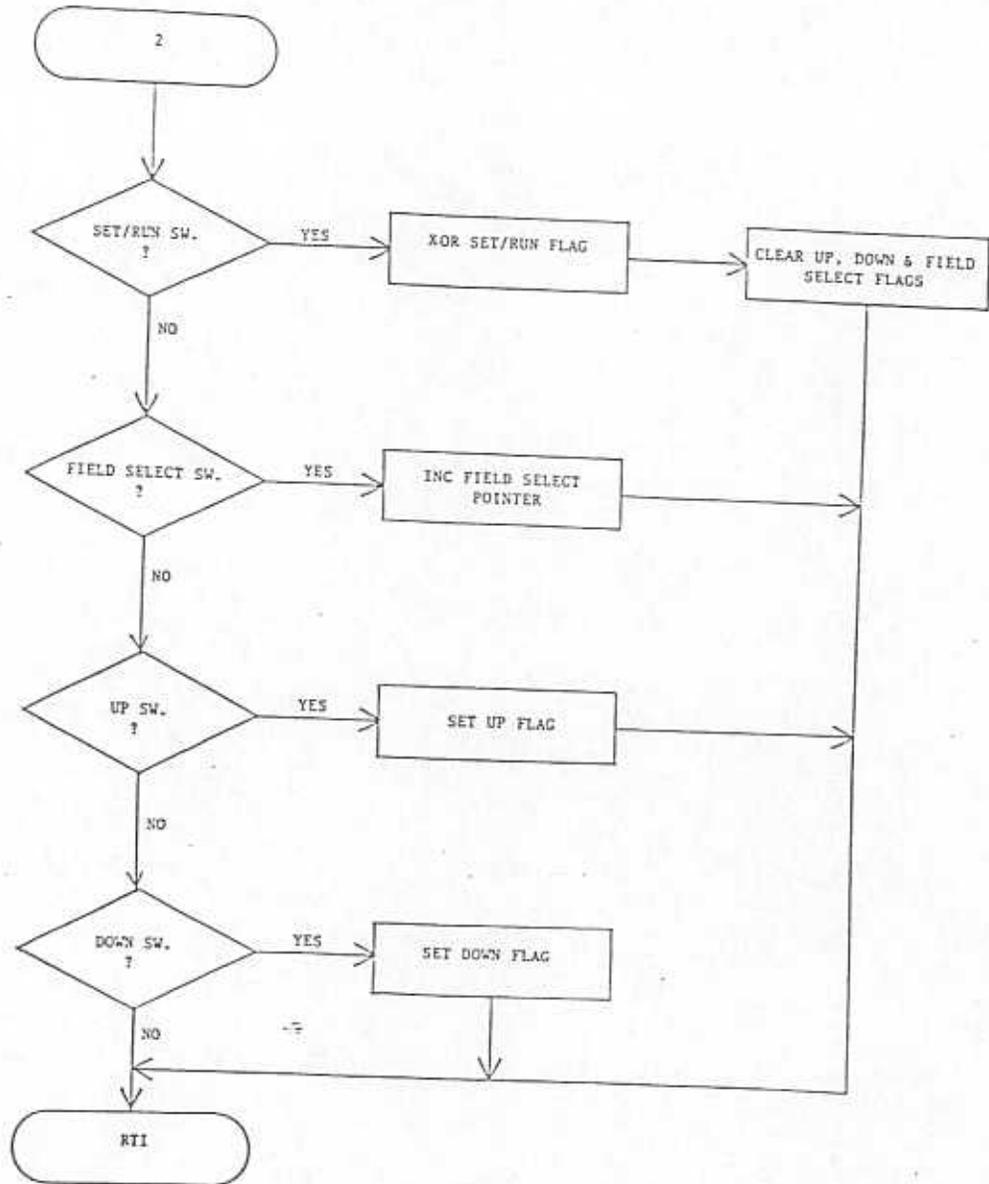


Figure 34

FRONT PANEL SWITCH HANDLER - PART 3

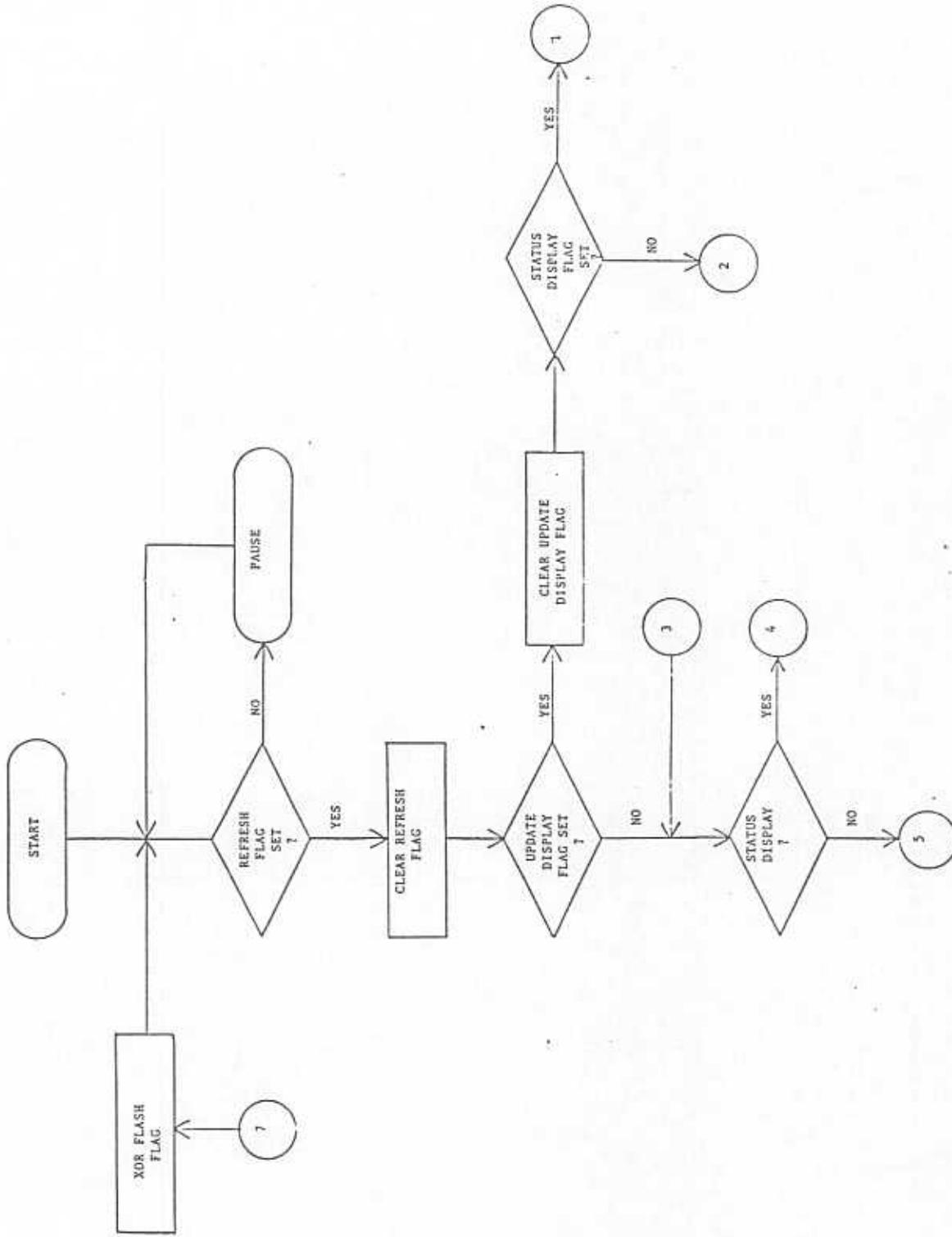


Figure 35

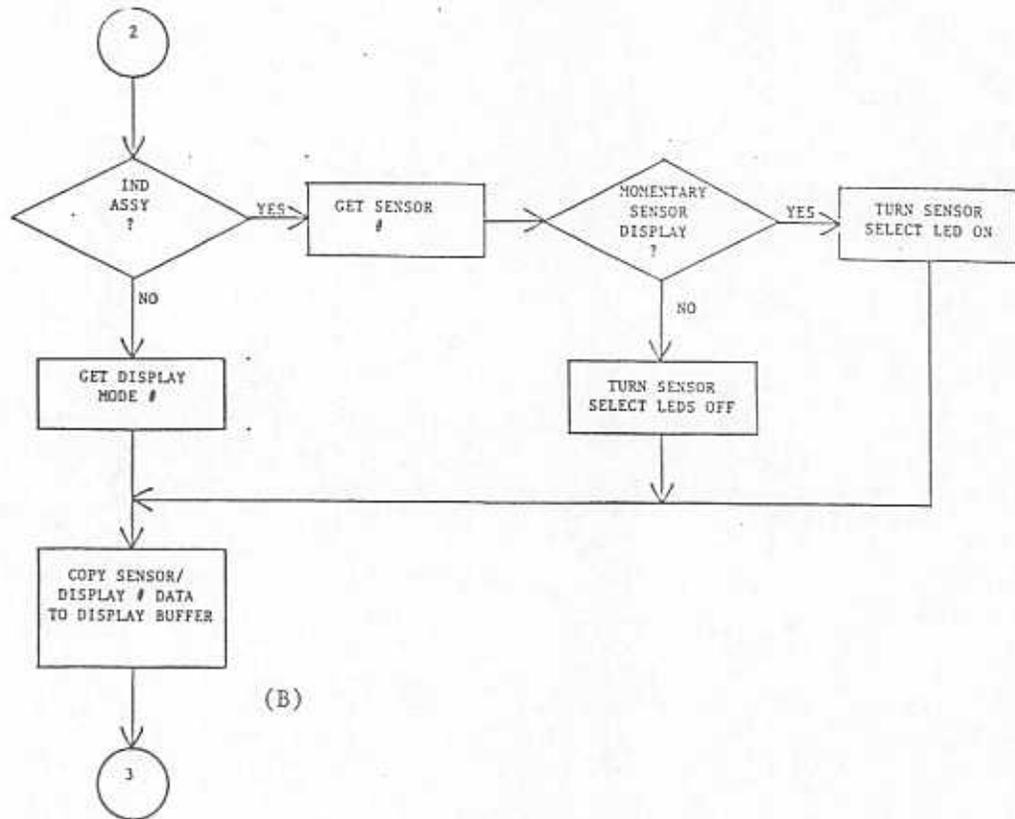
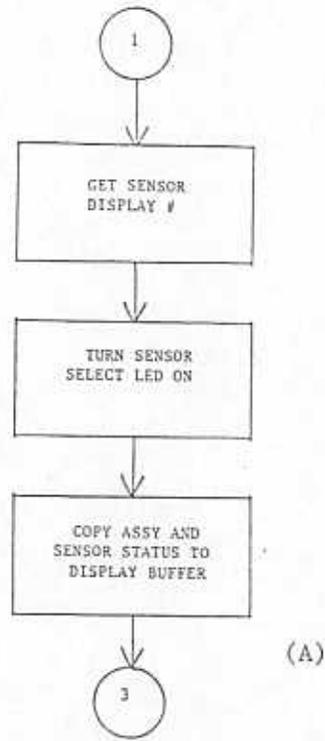


Figure 36 A & B

FRONT PANEL DISPLAY TASK - PART 2 A & B

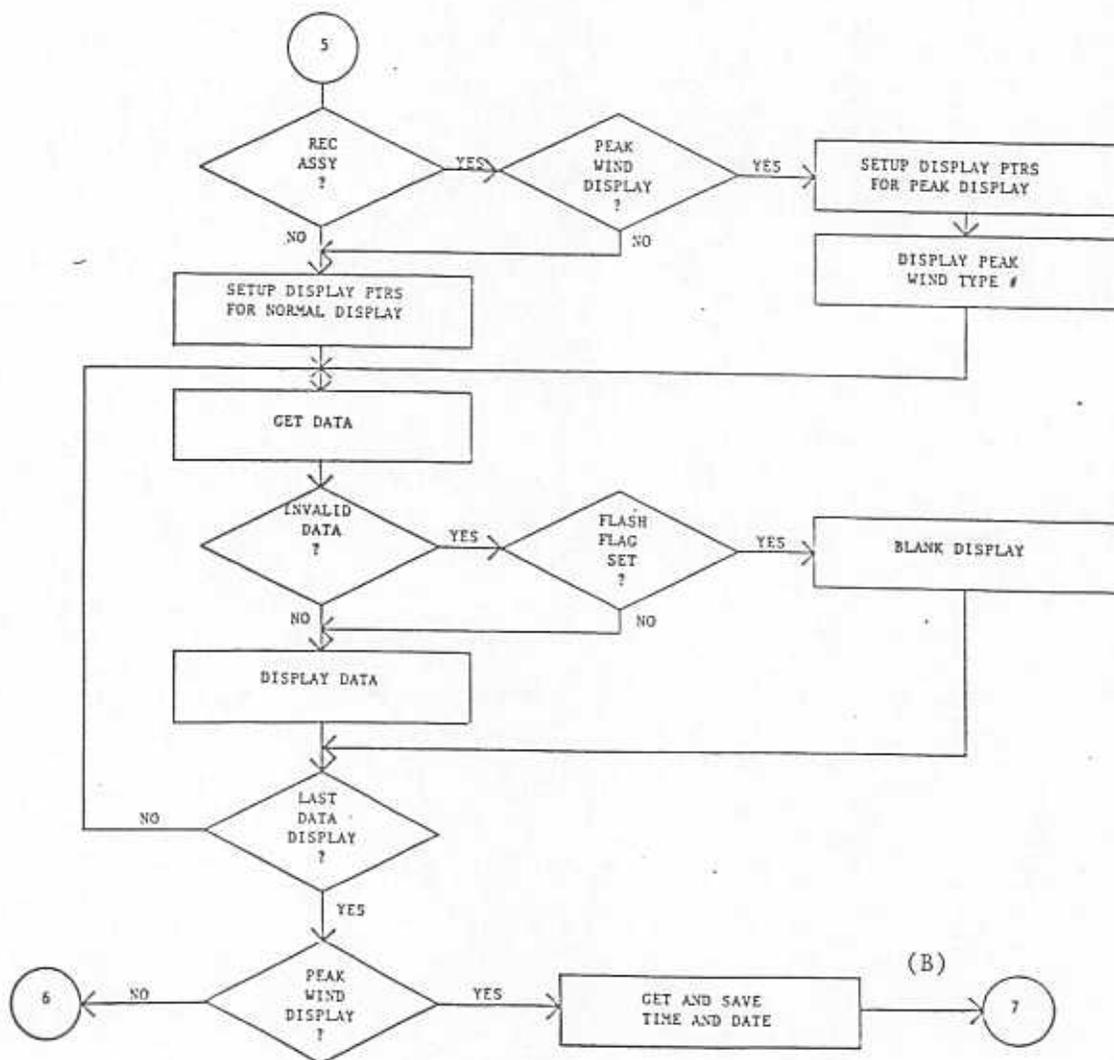
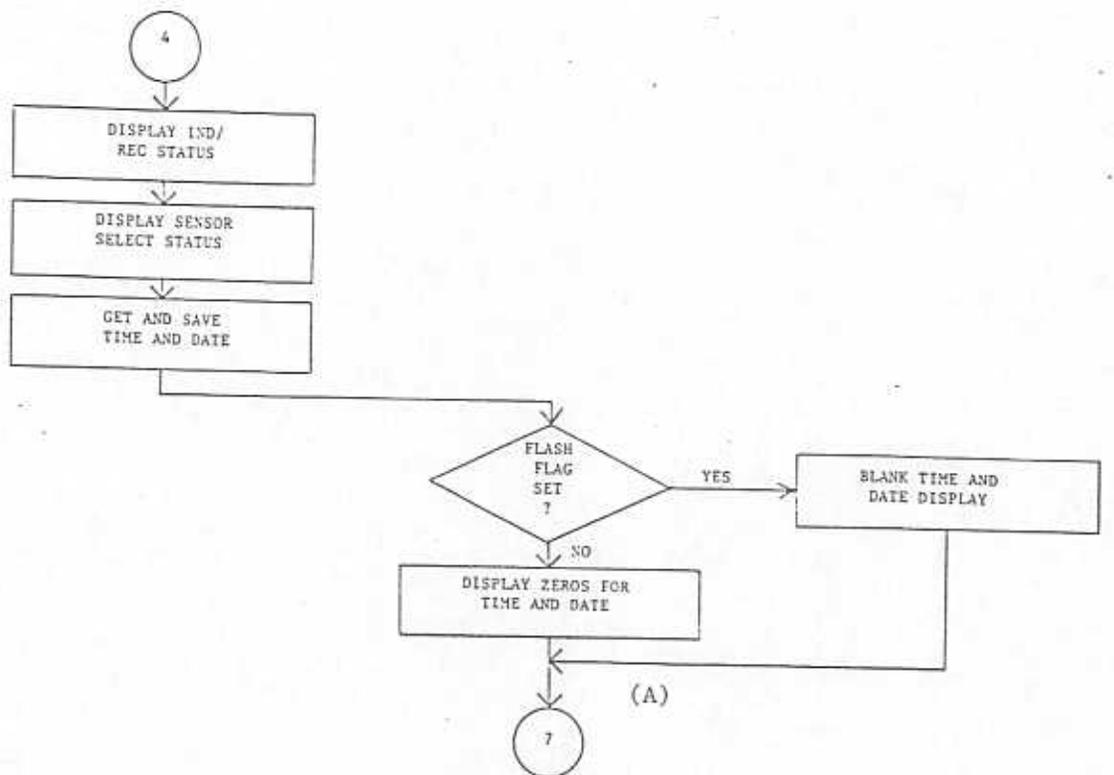


Figure 37 A & B

FRONT PANEL DISPLAY TASK - PART 3 A & B.

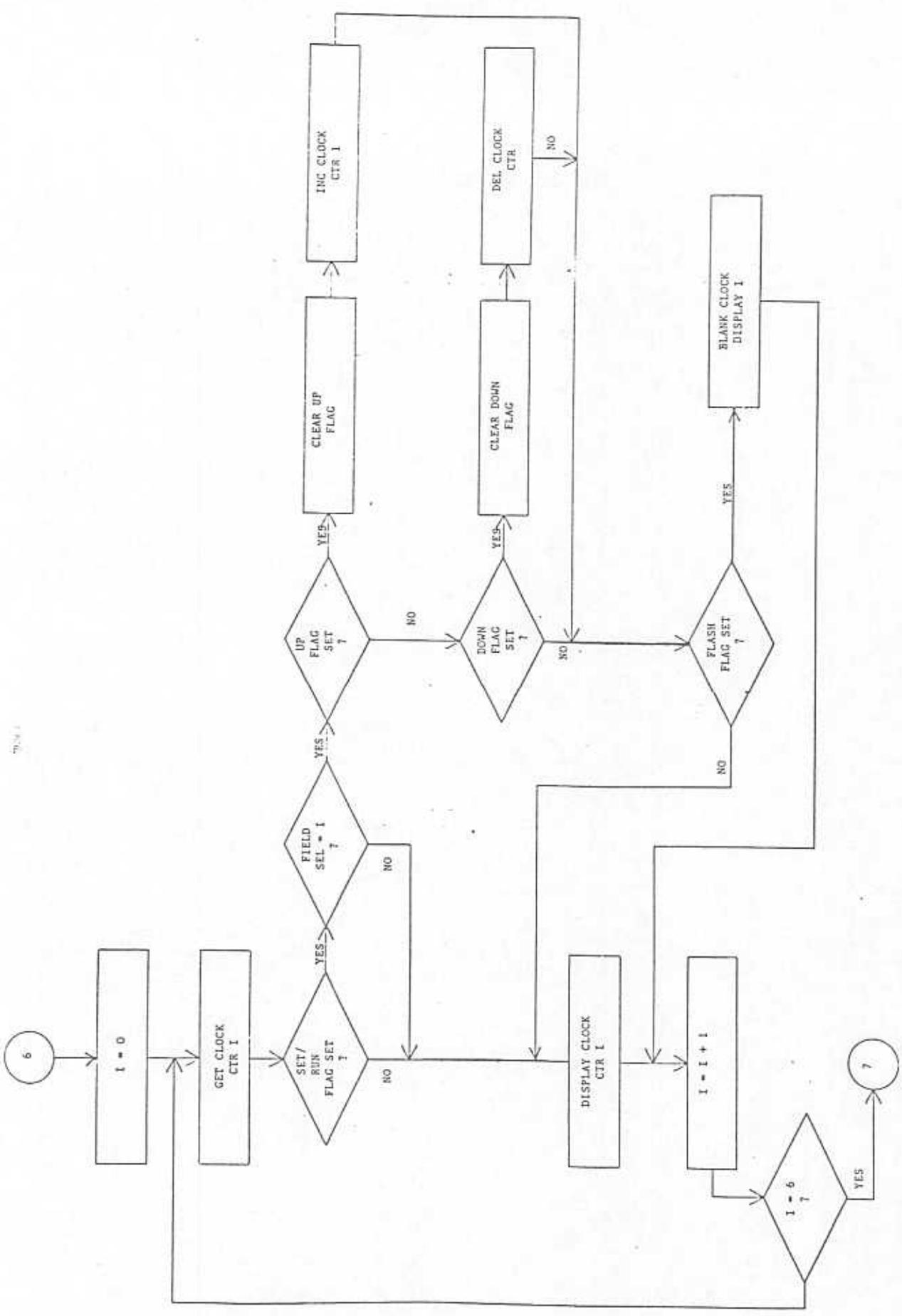


Figure 38

# AWDS CONTROL ROUTINE

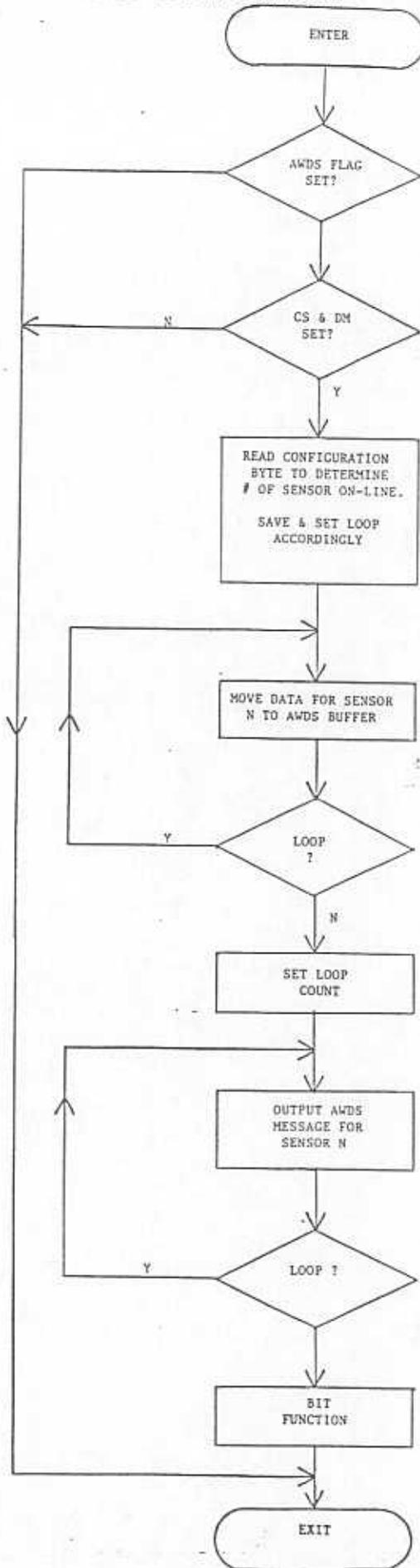


FIGURE 40  
AWDS OUTPUT ROUTINE

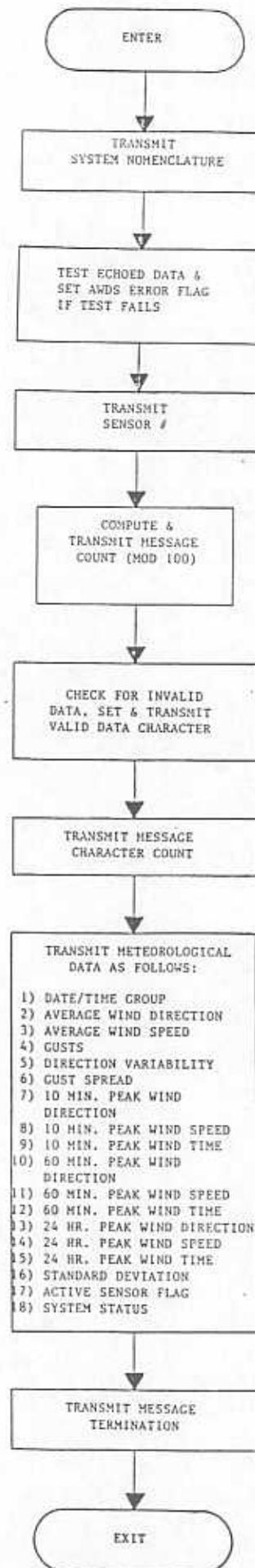


FIGURE 41  
PRINTER CONTROL ROUTINE

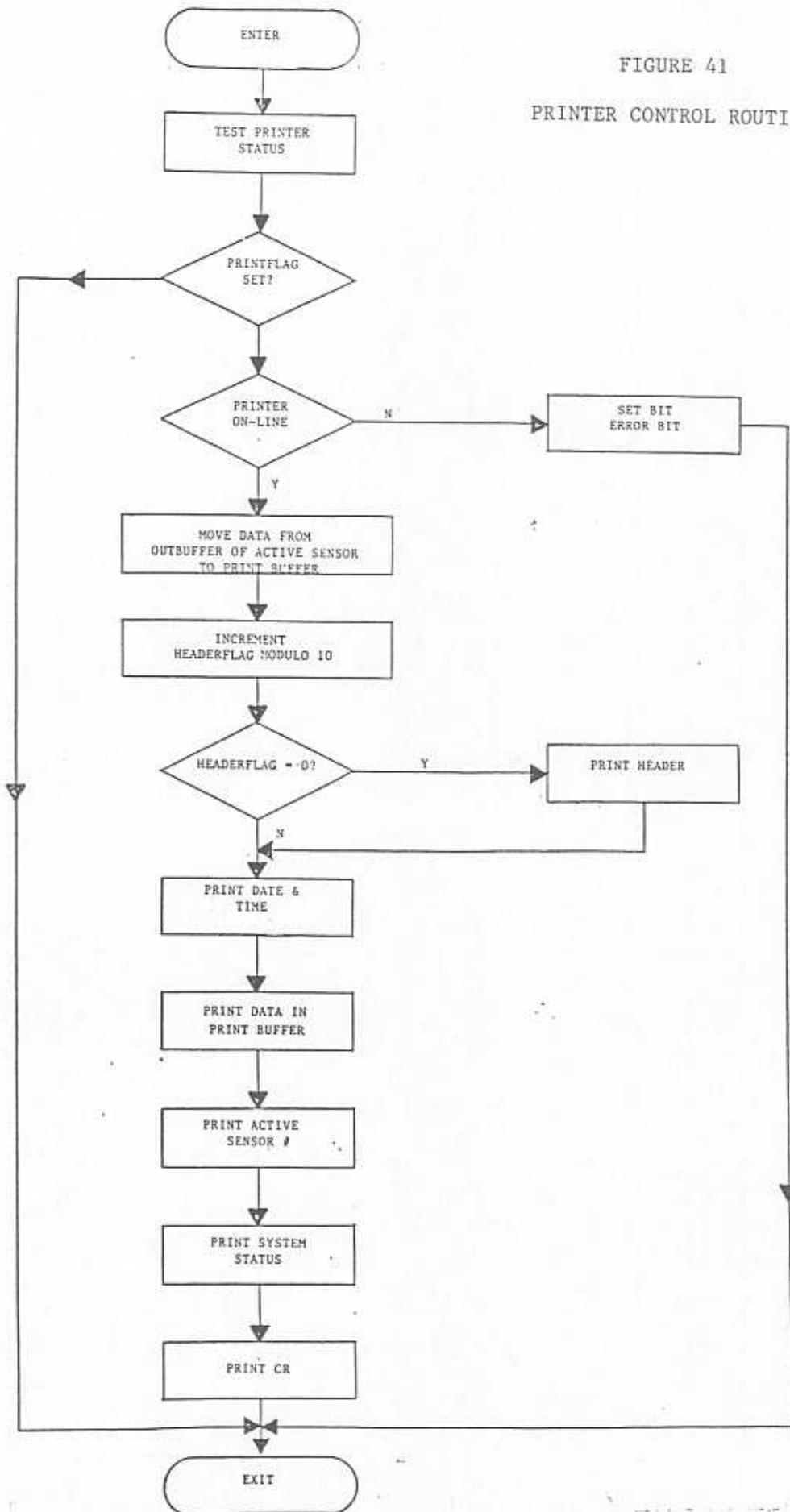
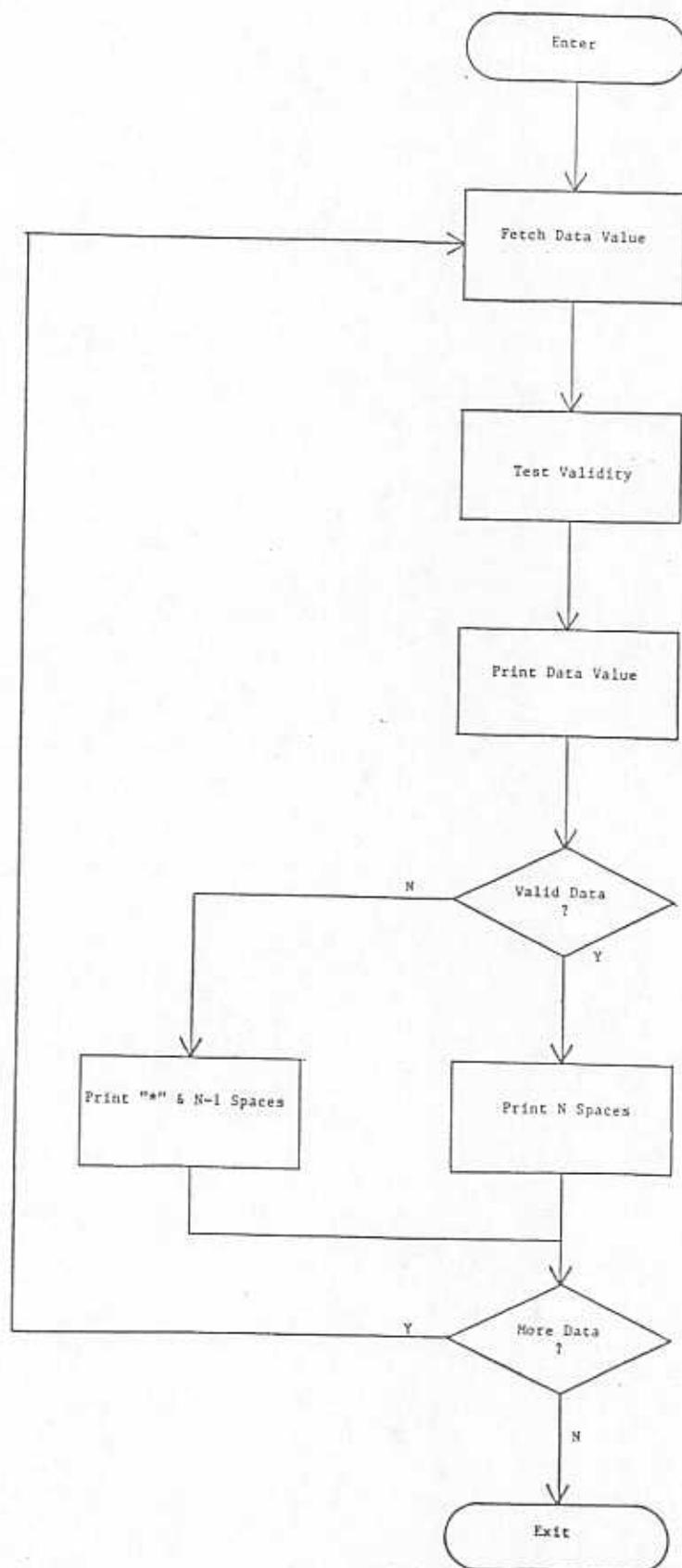


FIGURE 42  
PRINTER OUTPUT ROUTINE



INDICATOR/RECORDER STATUS WORD

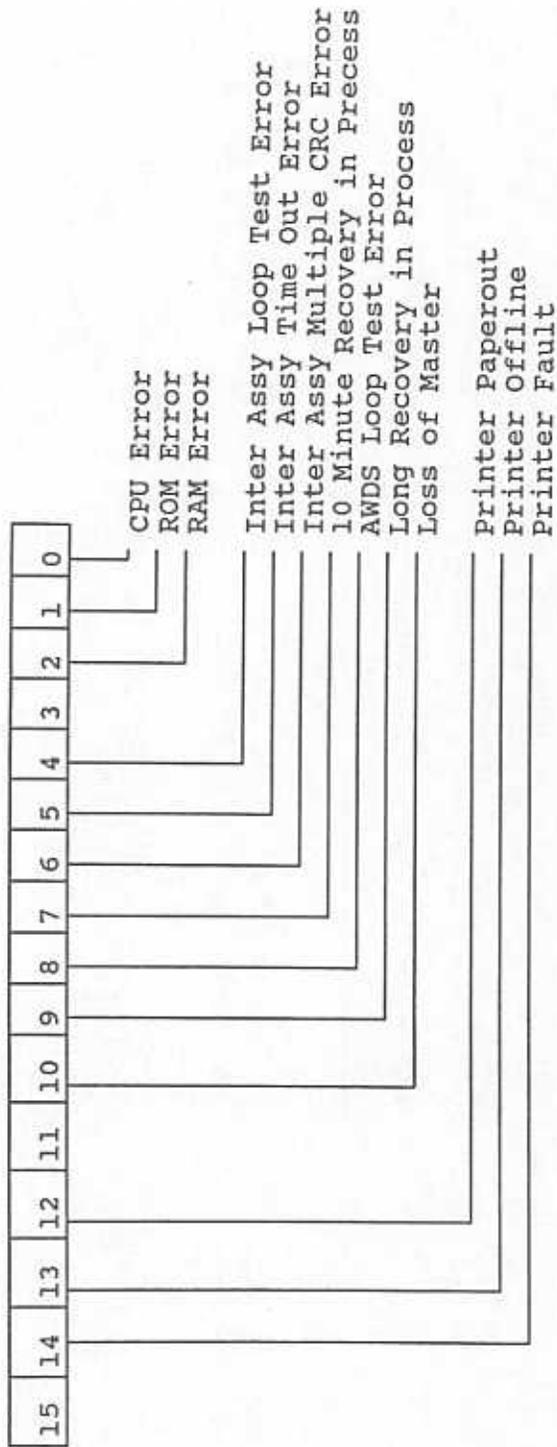


FIGURE 43 (A)

SENSOR STATUS WORD

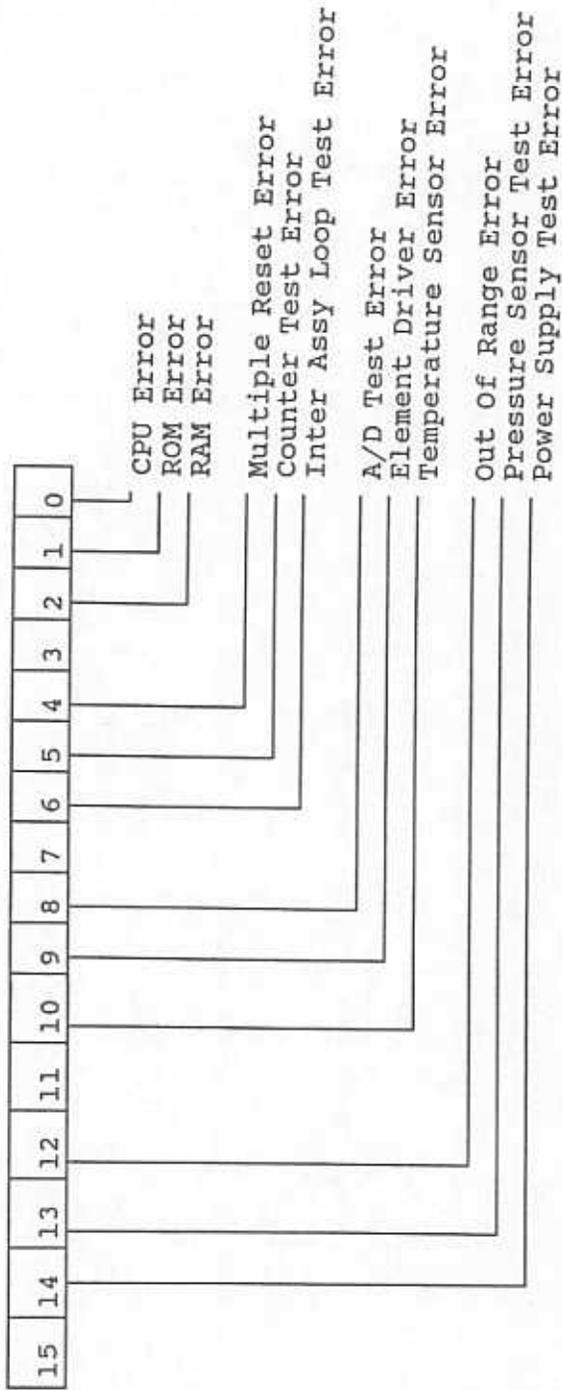
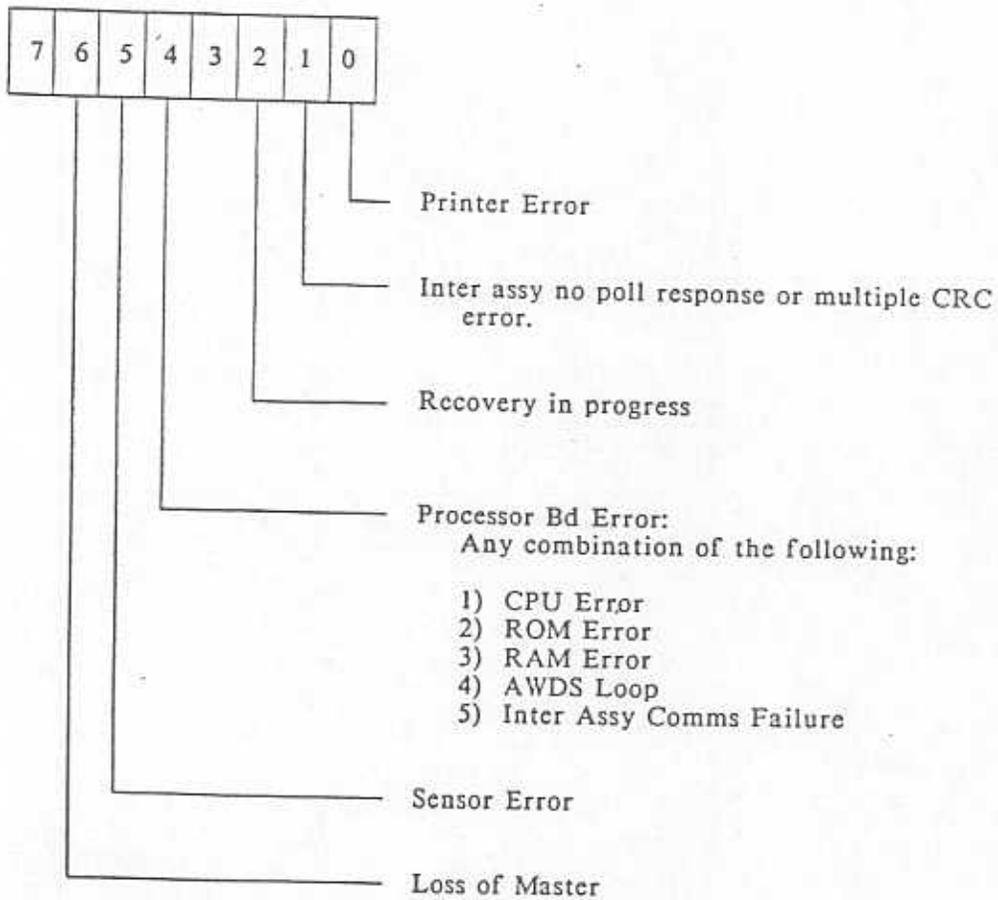


FIGURE 43 (B)

FIGURE 44  
GENERAL STATUS BYTE



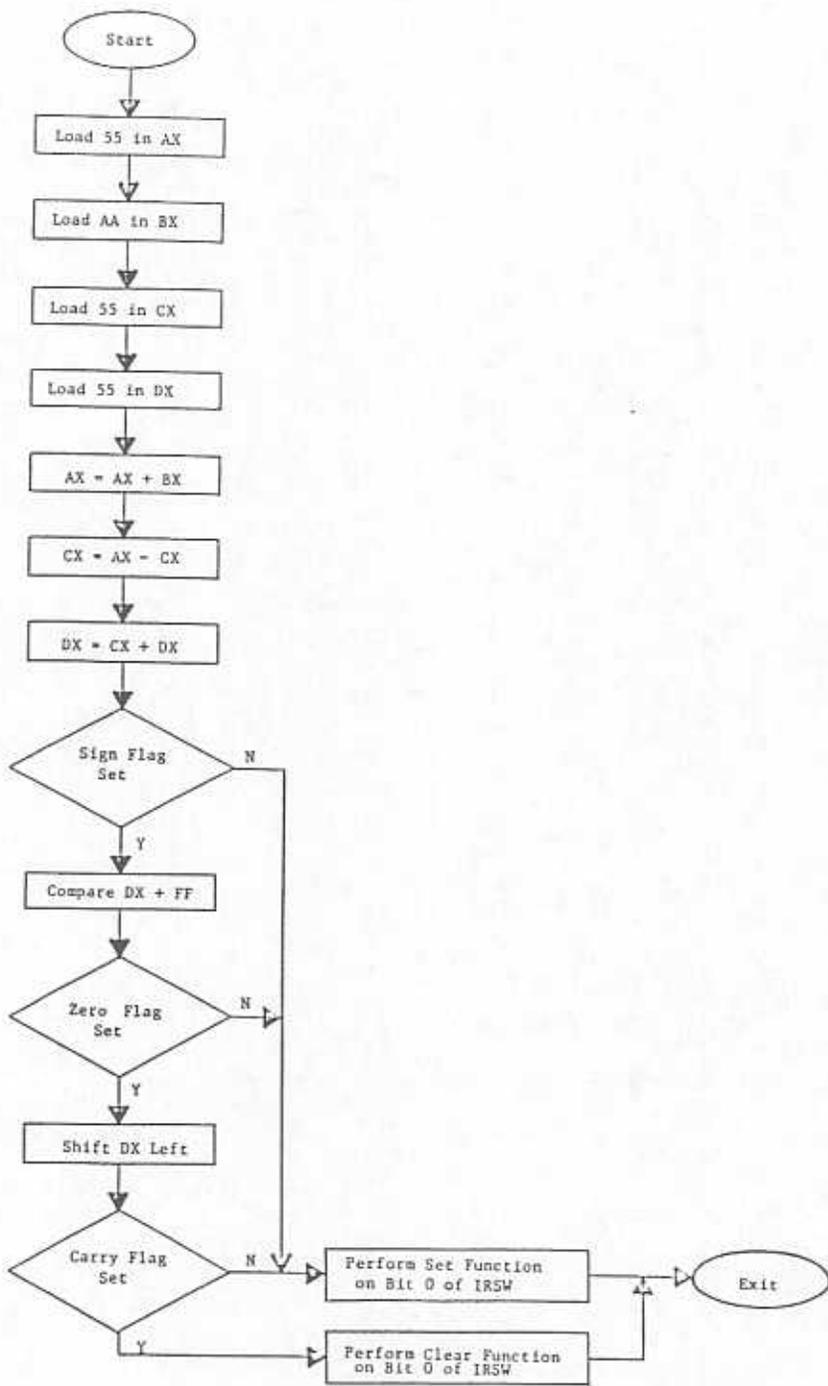


FIGURE 45

CPU TEST FLOW GRAPH

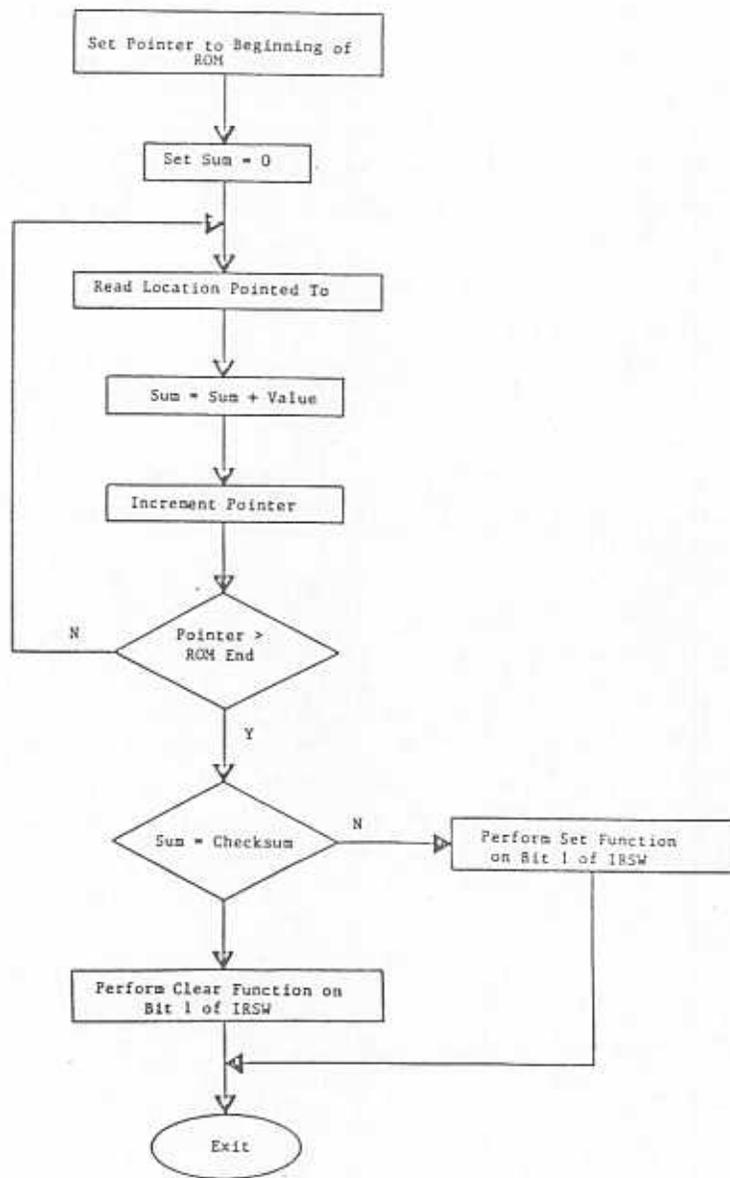
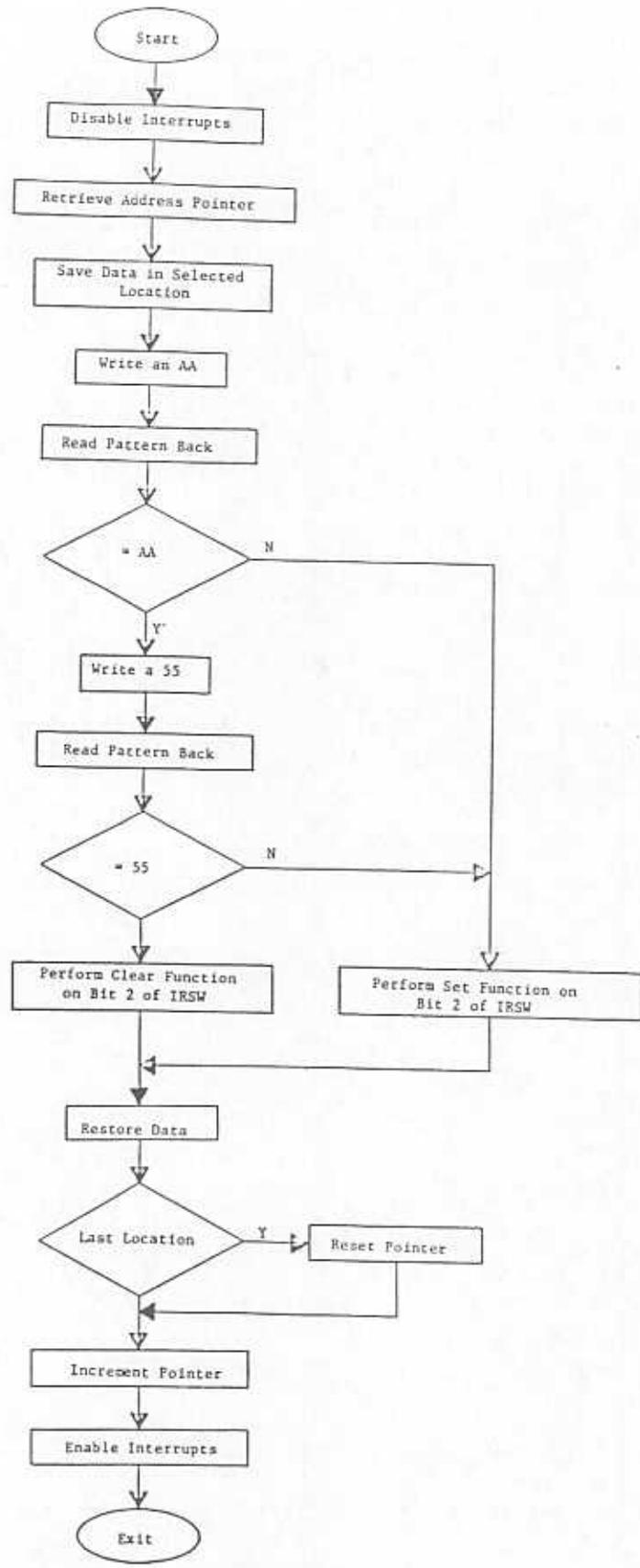


FIGURE 46

ROM TEST FLOW GRAPH

FIGURE 47  
RAM TEST FLOW GRAPH



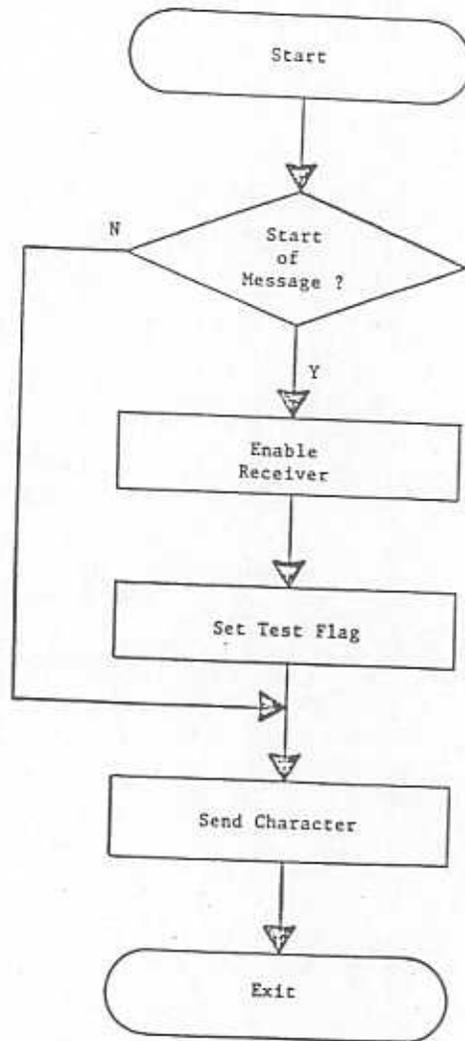


Figure 48A

IAC LOOP TEST TRANSMIT ROUTINE

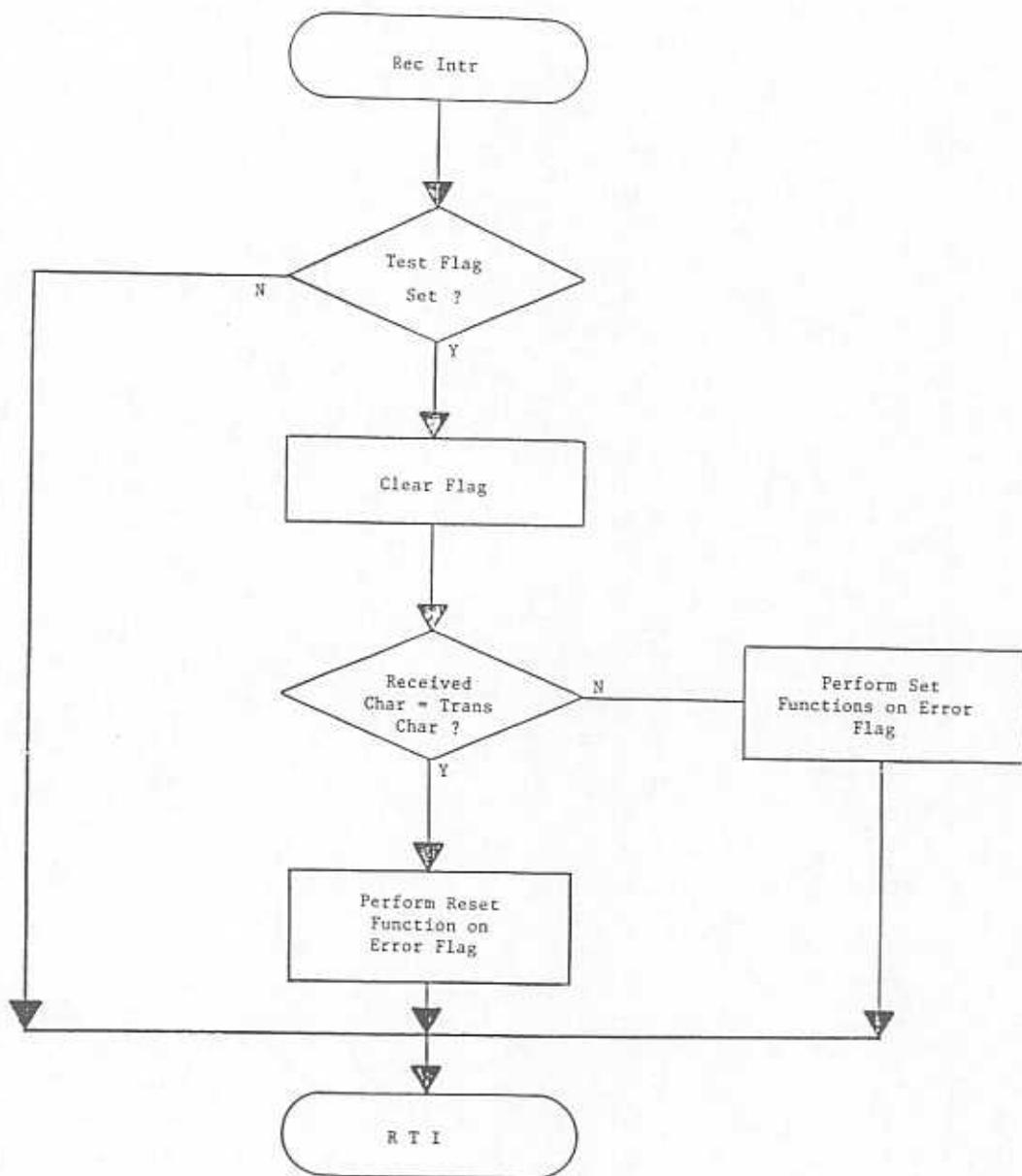


Figure 48B

IAC LOOP TEST RECEIVE ROUTING

10-10-70 10:01 10:01 10:01 10:01

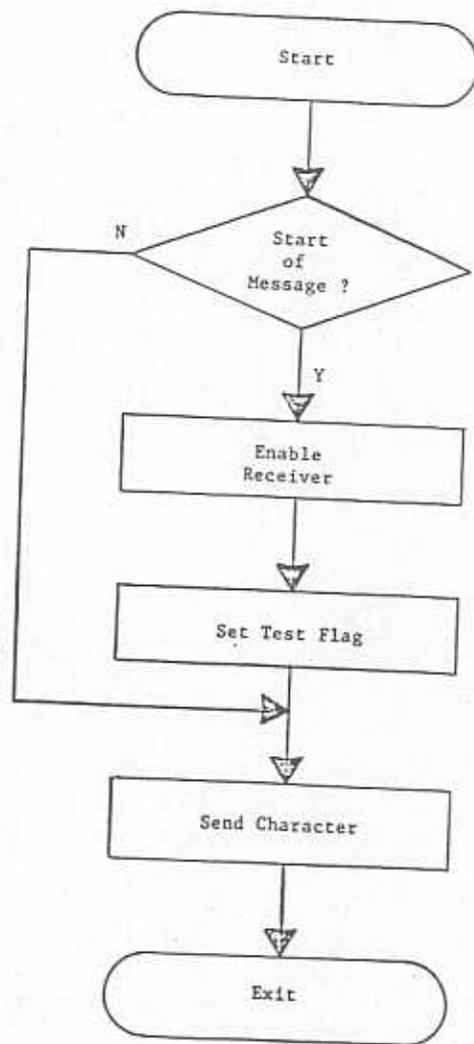


Figure 49A

AWDS LOOP TEST TRANSMIT ROUTINE

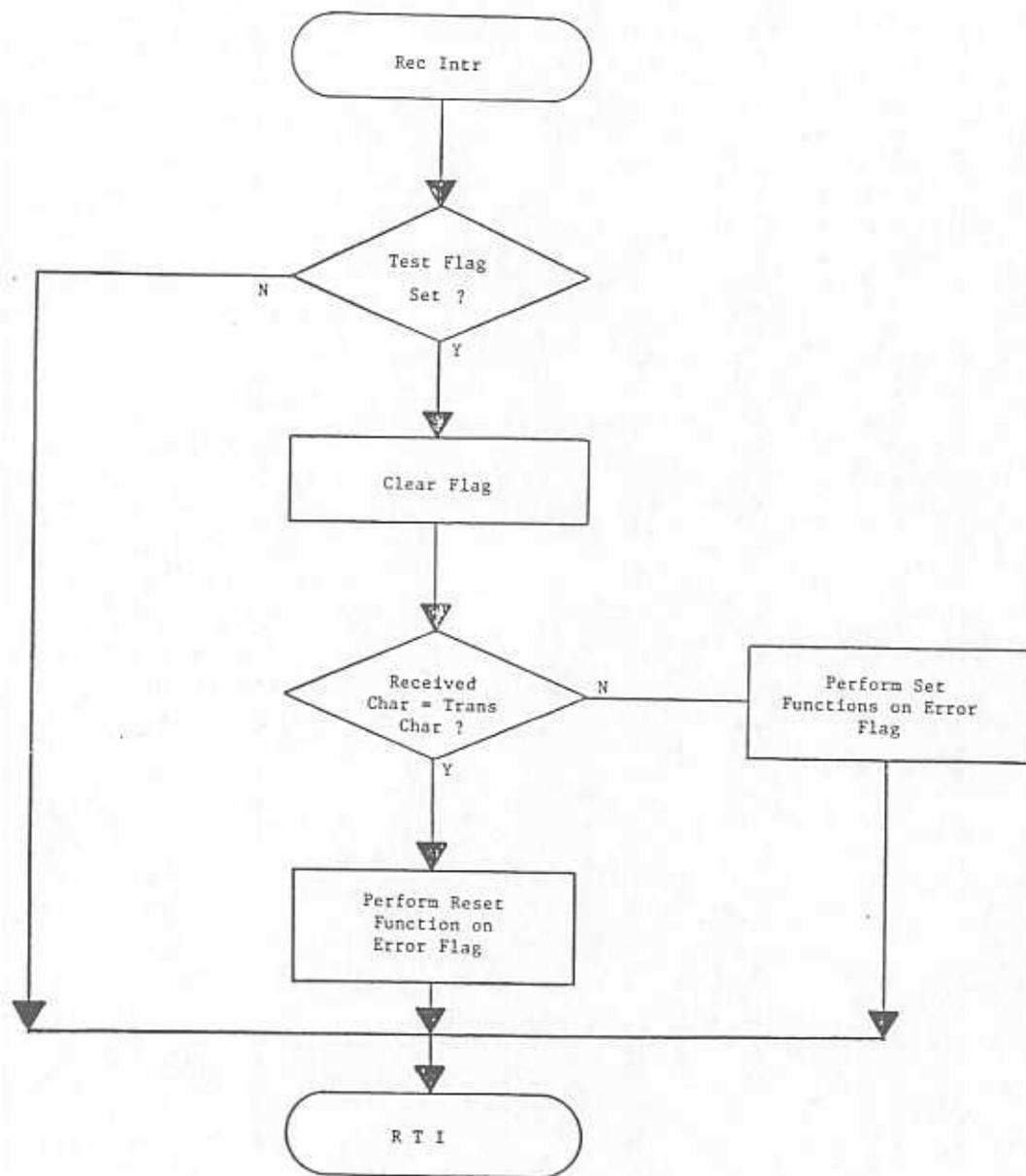


Figure 49B

AWDS LOOP TEST RECEIVE ROUTINE

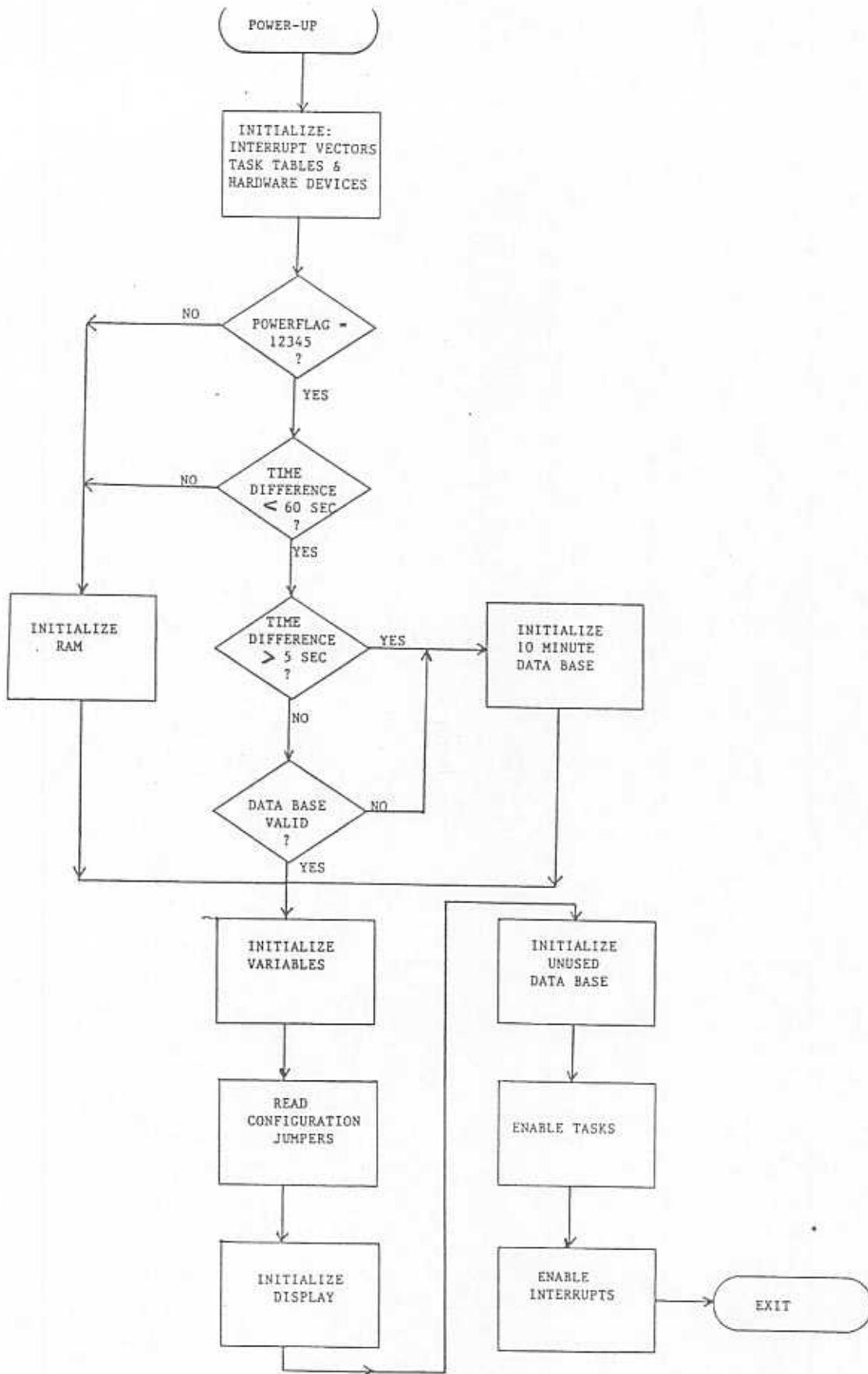
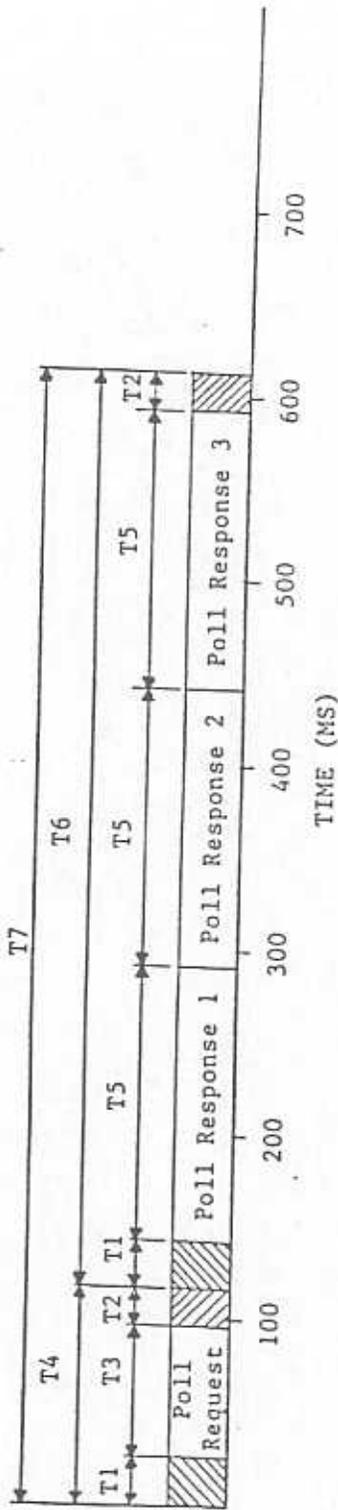


Figure 50 - Power-Up and Recovery Routine

POLLING TIMING



- T1 - Carrier Detect Off-On Delay - 25 MS
- T2 - Carrier Detect On-Off Delay - 20 MS
- T3 - Poll Request Message Period - 70 MS
- T4 - Poll Request Transmission Period - 115 MS
- T5 - Poll Response Message Period - 150 MS
- T6 - Poll Response Transmission Period - 495 MS
- T7 - Poll Session Period - 610 MS

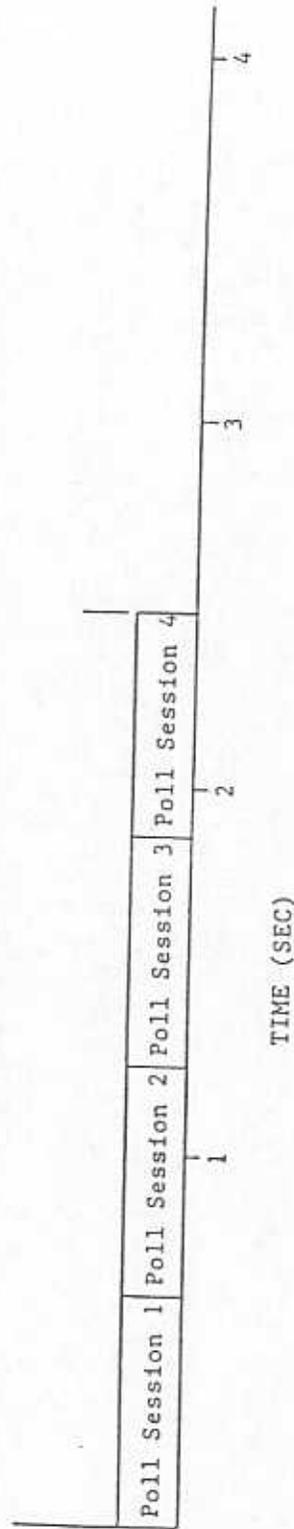


Figure 51 - Poll Timing



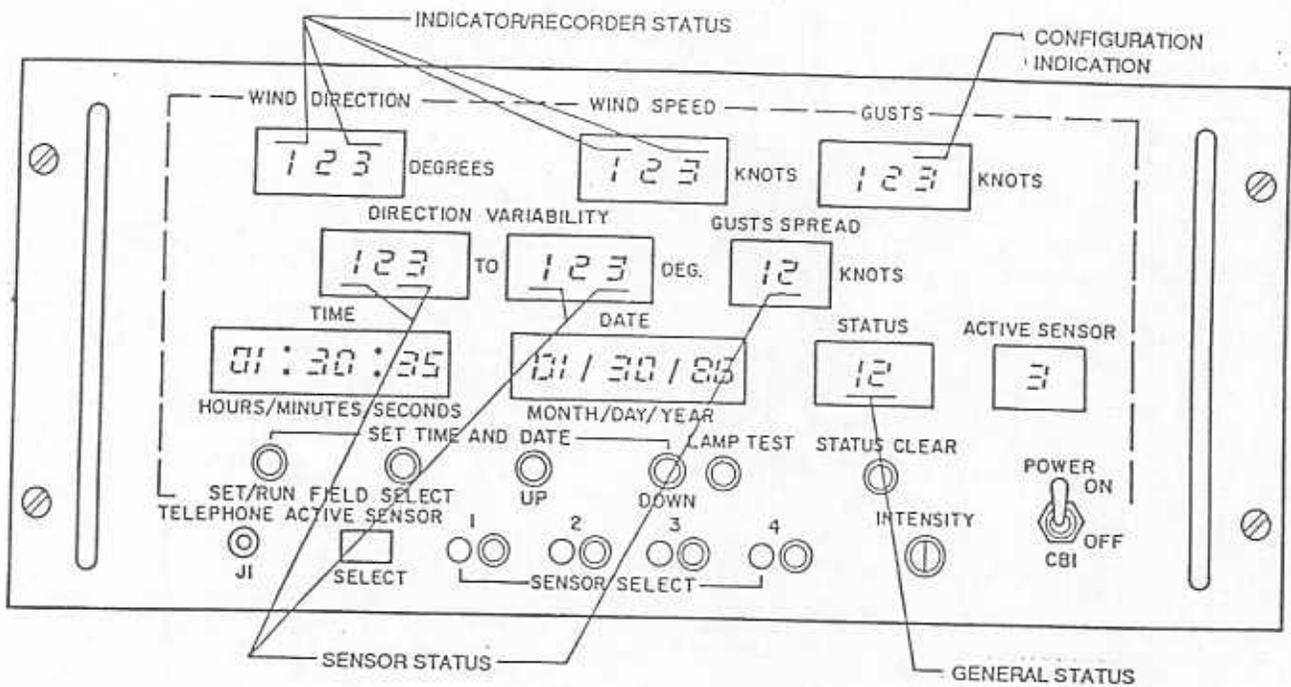


Figure 53(A) Error Code Display

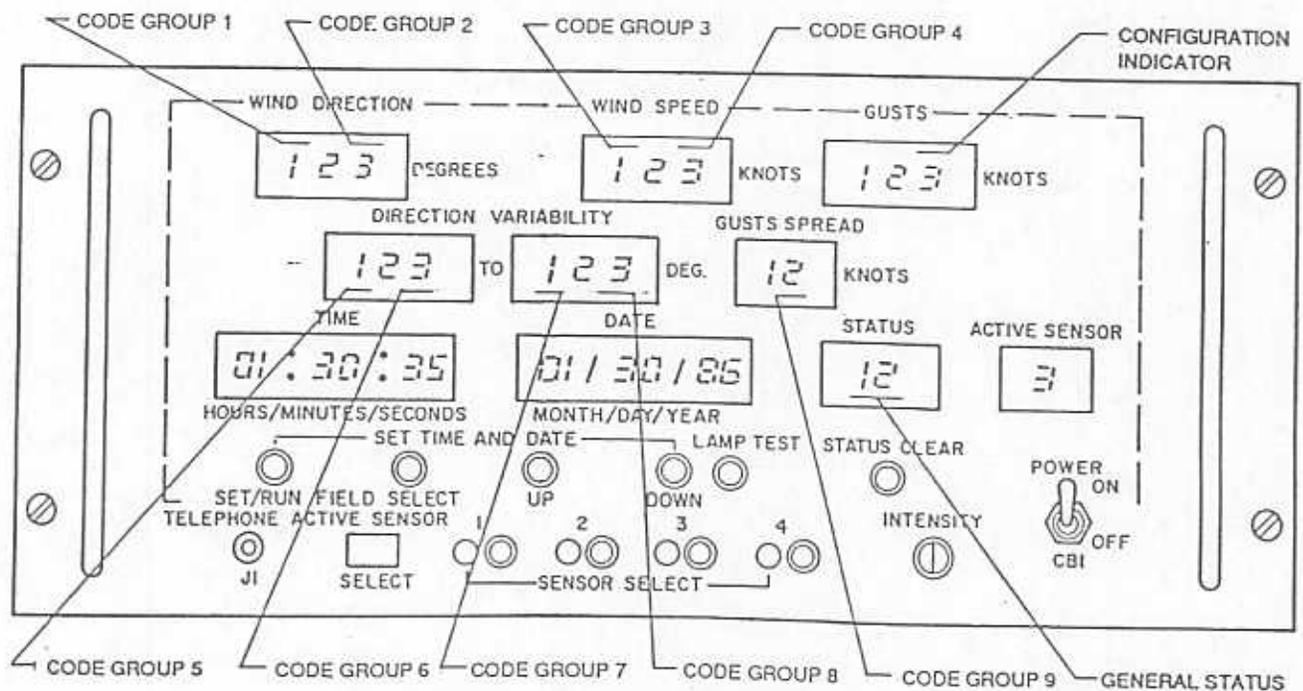


Figure 53(B) Error Code Groups

McClellan AFB  
Contract #: F04606-85-C0733  
CDRL Item: E00D  
15 May 90

---

---

---

**COMPUTER PROGRAM**

**SOURCE CODE LISTING**

**FOR THE**

**AN/FMQ-13(V)**

**DIGITAL INDICATOR/RECORDER**

**APPLICATION PROGRAM**

## 321 LIST

```

0 ( INDICATOR/RECORDER MEMORY ALLOCATION )
1
2 HEX 1F1F WIDTH !           ( SOURCE COMPILE W/ 31 CHAR NAME )
3
4 FFFF EQU /PROM             ( LAST ADDR. OF EPROM [FFFF] )
5
6 0000 EQU ORAM              ( 1ST ADDR. OF RAM [0000] )
7 1FFF EQU TOPRAM           ( LAST ADDR. OF RAM [1FFF] )
8
9 0000 EQU ZERO              ( SEGMENT REGISTERS OFFSET )
10
11 2 DICTIONARY              ( SYSTEM TARGET COMPILATION DICT. )
12
13 /PROM 3FFF - WINDOW       ( 1ST EPROM COMPIL. ADDR. [C000] )
14 40 ALLOT                  ( 1ST RAM COMPILATION ADDR. [0000] )
15 .S

```

## 322 LIST

```

0 ( INDICATOR/RECORDER TASKS RAM ALLOCATION )
1
2 HEX
3 TOPRAM 100 - EQU 'OPERATOR   ( DISPLAY TASK AREA [1EFF] )
4
5 TOPRAM 200 - EQU 'COMM.TASK  ( INTER-ASSY TASK AREA [1DFF] )
6
7 TOPRAM 300 - EQU 'AWDS.TASK  ( AWDS TASK AREA [1CFF] )
8
9 TOPRAM 400 - EQU 'WDOG       ( SYS. TEST TASK AREA [1BFF] )
10
11 TOPRAM 500 - EQU 'DATA.PROC  ( PROCESS TASK AREA [1AFF] )
12
13 TOPRAM 600 - EQU 'PRINT.TASK ( PRINTER TASK AREA [19FF] )
14
15 .S

```

## 323 LIST

```

0 ( RELEASE AND CHANGE HISTORY )
1 ( INITIAL VERSION RELEASED 11/16/87 )
2 ( no.op ROUTINE ADDED FOR DEBUGGING )
3 ( BLK 421, L7 - SCREEN FORMAT CHANGED FOR READABILITY )
4 ( BLK 429, L10 - STD. DEV. PRINT FIELD INCREASED FROM 2 TO 3 )
5 ( BLK 488 - GUST ROUTINE CHANGED TO LIMIT TO 10 MINUTE PERIOD )
6 ( BLK 622 - ROUTINE BAD.SUM ADDED )
7 ( BLK 624 - L11 - CHANGED C! TO ! FOR POWERFLAG )
8 ( BLK 625, L5 - ADDED CALL TO BAD.SUM )
9 ( BLK 626, L5 - ADDED CALL TO BAD.SUM )
10 ( RELEASED AS VERSION A 6/2/88 )
11 ( BLK 327, L12 - EXPANDED LOAD TO SCREEN 569 )
12 ( BLK 332, L10 - ADDED VARIABLE 6SECCTR )
13 ( BLK 330, L9 - INCREASED MESSAGE DATA COUNT FROM 89 TO 90 )
14 ( BLK 471, L6 - CHANGED ROUTINE MAX.V TO USE WHOULE KNOTS )
15 ( BLK 474, L11 - DROPPED SCALE.DOWN FROM PK.WD TO MATCH MAX.V )

```

324 LIST

- 0 ( RELEASE AND CHANGE HISTORY - CONTINUED )
- 1 ( BLK 481, L0 - CHANGED NAME OF WORD FROM MAX.V TO MAX.P )
- 2 ( BLK 569 - ADDED 10 SEC. DELAY ROUTINE FOR PRINTOUT @ 55 PAST)
- 3 ( BLK 570, L1 - ADDED VARIABLE 6SECCTR )

---

- ~~4 ( BLK 572 & 573 - REARRANGED LINE BETWEEN BLOCKS )~~

---

- 5 ( RELEASED AS VERSION B 5/10/89 )

---

- 6 ( BLK 473, L7 - LIMITED GUSTSPREAD TO 99 KNOTS )
- 7 ( RELEASED AS VERSION C 8/11/89 ) .S
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

325 LIST

- 0 ( McCLELLAN INDICATOR/RECORDER LOAD SCREEN 8/8/86)
- 1 ( REVISION C 8-11-89 )
- 2 HEX 1F1F WIDTH ! DECIMAL
- 3
- 4 330 351 THRU ( VARIABLES & CONSTANTS)
- 5
- 6 352 357 THRU ( SYSTEM ROUTINES )
- 7
- 8 360 367 THRU ( REAL TIME CLOCK ROUTINES )
- 9
- 10 370 403 THRU ( DISPLAY TASK ROUTINES )
- 11
- 12 326 328 THRU
- 13
- 14 .S
- 15

326 LIST

- 0 ( McCLELLAN INDICATOR/RECORDER LOAD SCREEN 8/8/86)
- 1
- 2 405 411 THRU ( SYSTEM TEST & WATCHDOG TASK )
- 3
- 4 413 421 THRU ( PRINTER TASK ROUTINES )
- 5
- 6 425 431 THRU ( AWDS TASK ROUTINES )
- 7
- 8 441 442 THRU ( WIND DATA - DATABASE ROUTINES)
- 9
- 10 445 450 THRU ( MATH & TRIG ROUTINES)
- 11
- 12 454 465 THRU ( AVERAGE DIRECTION & SPEED ROUTINES)
- 13 .S
- 14
- 15

## 327 LIST

0	( McCLELLAN INDICATOR/RECORDER LOAD SCREEN	8/8/86)
1		
2	470 482 THRU	( PEAKWIND ROUTINES )
3		
4	<del>485 488 THRU</del>	<del>( GUST ROUTINES )</del>
5		
6	490 494 THRU	( DIRECTION VARIABILITY ROUTINES )
7		
8	498 527 THRU	( STD. DEV. & PROCESSING TASK ROUTINES )
9		
10	530 562 THRU	( INTER-ASSEMBLY TASK ROUTINES )
11		
12	569 574 THRU	( RTC INTERRUPT HANDLER)
13	.S	
14		
15		

## 328 LIST

0	( McCLELLAN INDICATOR/RECORDER LOAD SCREEN	8/8/86)
1		
2	576 584 THRU	( SWITCH INTERRUPT HANDLER)
3		
4	587 598 THRU	( INTERRUPT HANDLERS)
5		
6	605 616 THRU	( SYSTEM & DEVICE INITIALIZATION)
7		
8	619 621 THRU	( TASK INITIALIZATION)
9		
10	622 631 THRU	( START-UP ROUTINE )
11		
12	635 636 THRU	( POWER-UP ROUTINE )
13	.S	
14		
15		

## 329 LIST

0	( INTENTIONALLY BLANK SCREEN )
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

## 330 LIST

```

0 ( INDICATOR/RECORDER SYSTEM VARIABLES 7/7/86)
1
2 THERE CONSTANT RAMSTART ( START OF SYSTEM VARIABLES )
3
4 CVARIABLE 55SECCTR ( 5 SECONDS COUNTER )
5 CVARIABLE 500MCTR ( 500 mSEC. COUNTER )
6 CVARIABLE SENSOR# ( ACTIVE & MOMENTARY SENSOR # )
7 CVARIABLE STATBYTE ( SYSTEM STATUS BYTE)
8 CVARIABLE EXTSTATFLG ( UART A EXTERNAL STATUS )
9 CVARIABLE CONFIG ( ASSEMBLY CONFIGURATION FLAG )
10 CVARIABLE LEAP ( LEAP YEAR FLAG)
11 VARIABLE 1stSTART 6 ALLOT ( FLAG TO INDICATE FIRST PROCESS )
12 VARIABLE POWERFLAG ( FLAG TO INDICATE FIRST POWER-UP)
13
14 .S
15

```

## 331 LIST

```

0 ( INDICATOR/RECORDER BUILT IN TEST STATUS WORDS 1/2/87 )
1
2 VARIABLE I/RSTATUS ( IND/REC STATUS WORD)
3 VARIABLE SENSTAT 6 ALLOT ( SENSORS 1-4 STATUS WORDS )
4 VARIABLE I/RACK ( STATUS ACKNOWLEDGE WORD)
5 VARIABLE SENACK 6 ALLOT ( SENSORS 1-4 STATUS ACK WORDS)
6
7 ( *** DO NOT SEPERATE THE ABOVE VARIABLES *** )
8
9
10 .S
11
12
13
14
15

```

## 332 LIST

```

0 ( INDICATOR/RECORDER DISPLAY TASK VARIABLES 5/3/89)
1
2 CVARIABLE DISPFLAG ( 5 SEC UPDATE DISPLAY FLAG )
3 CVARIABLE REFRESHFLAG ( 500 mSEC DISPLAY REFRESH FLAG )
4 CVARIABLE LEDSTATE ( FPI DATA ERROR FLAGS)
5 CVARIABLE STATDISFLAG ( STATUS DISPLAY FLAG)
6 CVARIABLE 1MINCTR ( 1 MIN. MOMENT. DISPLAY CTR )
7 CVARIABLE STDFLAG ( 1 MIN STD DEV UPDATE FLAG )
8 CVARIABLE 55FLAG ( HR & DAY PEAK WIND UPDATE FLAG)
9 CVARIABLE CLOCKFLAG ( 8-BITS FLAG FOR CLOCK & FLASH )
10 CVARIABLE 6SECCTR ( DELAY CTR FOR 55 MIN. UPDATE )
11 .S
12
13
14
15

```

## 333 LIST

```

0 ( INDICATOR/RECORDER  AWDS & PRINTER TASK VARIABLES      7/7/86)
1
2 CARIABLE AWDSFLAG          ( 5 SEC AWDS OUTPUT FLAG )
3 CARIABLE MESSAGE#         ( MOD 99 COUNTER FOR AWDS)
4 CARIABLE UARTRFLG         ( TEST UART B LOOPBACK FLAG )
5 CARIABLE CHARBUFFB        ( TEMP BUFFER FOR LOOPBACK CHAR )
6 CARIABLE UARTRERR         ( UART B LOOPBACK ERROR CTR )
7
8
9 CARIABLE PRINTFLAG        ( 1 MIN PRINTER OUTPUT FLAG )
10 CARIABLE HEADERFLAG      ( 1 MIN PRINTOUT HEADER FLAG )
11
12
13
14 .S
15

```

## 334 LIST

```

0 ( INDICATOR/RECORDER  INTER-ASSY TASK VARIABLES )
1
2 CARIABLE POLLFLAG          ( 5 SEC SENSOR POLLING FLAG )
3 CARIABLE TIMEOUT           ( COMM. TIMEOUT TIMER )
4 VARIABLE COMMBUFF 68 ALLOT ( COMM. I/O BUFFER )
5 VARIABLE STXPTR            ( STX CHAR POINTER )
6 VARIABLE ETXPTR            ( ETX CHAR POINTER )
7 CARIABLE POLLSEN#          ( TEMP. FOR SENSOR POLL # )
8 CARIABLE XMITFLAG          ( XMITTING POLL REQUEST FLAG )
9 CARIABLE UARTRFLG         ( TEST UART A LOOPBACK FLAG )
10 CARIABLE CHARBUFFA        ( TEMP BUFFER FOR LOOPBACK CHAR )
11 CARIABLE UARTRERR         ( UART A LOOPBACK ERROR CTR )
12
13 .S
14
15

```

## 335 LIST

```

0 ( INDICATOR/RECORDER  INTER-ASSY TASK VARIABLES )
1
2 CARIABLE CDFLAG            ( CARRIER DETECT FLAG )
3 CARIABLE CARTIMER          ( CARRIER TIMEOUT CTR )
4 CARIABLE ACTCTR            ( CARRIER DETECT COUNTER )
5 CARIABLE RESPFLAG          ( EXPECTING SENSOR RESP. FLAG )
6 CARIABLE POLLCNT           ( SENSOR POLLING COUNTER )
7 CARIABLE CONTRLFLAG        ( POLL CONTROL FLAG )
8 CARIABLE FAILCTR           ( TIMEOUT FAILURE COUNTER )
9 CARIABLE CRC16CNT          ( CRC-16 CKSUM ERROR CTR )
10 VARIABLE BADPOLLS 2 ALLOT ( FAIL POLL COUNTERS [4] )
11
12
13 .S
14
15

```

## 336 LIST

```

0 ( INDICATOR/RECORDER DEVICE ADDRESSES      7/7/86)
1
2 HEX
3 20 CONSTANT clock      ( BASE ADDRESS OF 7170 RTC)
4 30 CONSTANT clock/intx ( ADDRESS OF INTERRUPT MASKS/FLAGS)
5 31 CONSTANT clock/csr  ( ADDRESS OF CONTROL STATUS REG)
6
7 00 CONSTANT comm/data  ( INTER-ASSEMBLY COMMUNICATION PORT)
8 01 CONSTANT awds/data  ( AWDS COMMUNICATION PORT )
9 02 CONSTANT comm/csr   ( INTER-ASSEMBLY COMMUNICATION CONTROL)
10 03 CONSTANT awds/csr  ( AWDS COMMUNICATION CONTROL)
11
12
13 .S
14
15

```

## 337 LIST

```

0 ( INDICATOR/RECORDER DEVICE ADDRESSES      7/7/86)
1
2 HEX
3 40 CONSTANT watchdog
4
5 A0 CONSTANT fpi/data   ( FRONT PANEL INTERFACE PORT)
6 A1 CONSTANT fpi/csr    ( STATUS CONTROL REGISTER)
7
8 C0 CONSTANT print/reg  ( PRINTER OUTPUT PORT)
9 E0 CONSTANT mcs/reg    ( PRINTER STATUS, SW. LED, & ALARM REG)
10
11 .S
12
13
14
15

```

## 338 LIST

```

0 ( INDICATOR/RECORDER DISPLAY TASK CONSTANTS  7/7/86)
1
2 HEX
3 A4 CONSTANT ASIDE     ( FPI NIBBLE B INHIBIT COMMAND )
4 A8 CONSTANT BSIDE     ( FPI NIBBLE A INHIBIT COMMAND)
5 FFF CONSTANT BLANKIT ( LED BLANKING CODE)
6 .S
7
8
9
10
11
12
13
14
15

```

## 339 LIST

```

0 ( INDICATOR/RECORDER REAL TIME CLOCK CONSTANTS 7/7/86)
1
2 HEX
3 01 CONSTANT HOUR ( OFFSETS TO COUNTERS)
4 02 CONSTANT MINUTE 03 CONSTANT SEC
5 04 CONSTANT MONTH 05 CONSTANT DAY
6 06 CONSTANT YEAR
7 09 CONSTANT RAMHOUR
8 0A CONSTANT RAMMIN 0B CONSTANT RAMSEC
9 0C CONSTANT RAMMONTH 0D CONSTANT RAMDAY
10 0E CONSTANT RAMYEAR
11 .S
12
13
14
15

```

## 340 LIST

```

0 ( INDICATOR/RECORDER CLOCK HANDLER CONSTANTS 7/7/86)
1
2 HEX
3 365 4 * 1+ CONSTANT D4Y ( # OF DAYS IN 4 YEARS )
4
5 CREATE D/M ( NUMBER OF DAYS OF YEAR ACCORDING TO MONTH )
6 0 , 31 , 59 , 90 , 120 , 151 , 181 , 212 ,
7 243 , 273 , 304 , 334 ,
8 .S
9
10
11
12
13
14
15

```

## 341 LIST

```

0 ( WIND DATA VARIABLES AND CONSTANTS 3/11/86)
1
2 1210 CONSTANT FILESIZE ( FILESIZE INCORPORATES ALL QUEUES
3 AND VARIABLE FOR EACH SENSOR )
4
5 VARIABLE DATABASE FILESIZE 4 ( # OF SENSORS ) * ALLOT
6
7
8 ( VARIABLES WITHIN THE DATABASE ARE DEFINED ON THE NEXT BLOCK)
9
10
11 7967 WIDTH !
12 .S
13
14
15

```

## 342 LIST

```

0 ( WIND DATA OFFSET CONSTANTS FOR DATA BLOCKS          3/11/86)
1
2 ( THE SPEED QUEUE HAS AN OFFSET OF 0 & IS 240 BYTES LONG)
3 240 CONSTANT DIRQ ( OFFSET TO DIRECTION QUE, 240 BYTES)
4 480 CONSTANT GUSTQUE ( OFFSET TO GUST QUE, 240 BYTES)
5 720 CONSTANT PVDBDIR ( OFFSET TO PVD DIRECTION QUE, 48 BYTES)
6 768 CONSTANT PVDSPD ( OFFSET TO PVD SPEED QUE, 48 BYTES)
7 816 CONSTANT PVDTIM ( OFFSET TO PVD TIME QUE, 48 BYTES)
8 864 CONSTANT RESULTBUFFER ( OFFSET TO RESULT BUFFER, 32 BYTES )
9 896 CONSTANT PEAKRESULTBUF ( OFFSET TO PEAK RESULT BUFFER, 14 )
10 910 CONSTANT INBUF ( OFFSET TO INPUT BUFFER, 10 BYTES)
11 920 CONSTANT SCRATCH ( OFFSET TO SCRATCH PAD AREA, 38 BYTES)
12 960 CONSTANT TIMEQUE ( OFFSET TO PROCESS TIME AREA, 240 BYTES)
13 ( **** PRESENT FILESIZE IS 1200 BYTES 5/20/87 **** )
14
15 .S

```

## 343 LIST

```

0 ( WIND DATA VARIABLES          3/12/86)
1
2 VARIABLE #THETAS ( # OF NON-ZERO THETAS USED IN STD. DEV.)
3 VARIABLE TEMP ( TEMPORARY STORAGE )
4 VARIABLE MXVALUE ( TEMPORARY VALUE & RESULTANT MAX)
5 VARIABLE OFFSET ( INDEX POINTER USED TO RETURN DATA)
6 VARIABLE MNVALUE ( TEMPORARY VALUE AND RESULTANT MIN)
7 VARIABLE OUTPOINTER ( ADDRESS POINTER USED IN SORTING)
8 VARIABLE THETA0 ( DIR.-VAR. TEMP STORAGE )
9 VARIABLE DELTADelta ( DIR.-VAR. TEMP STORAGE )
10 VARIABLE SENSORSTATUS ( TEMPORY LOCATION FOR NEW STATUS)
11 .S
12
13
14
15

```

## 344 LIST

```

0 ( WIND DATA VARIABLES CONTINUED          4/8/86 )
1
2 VARIABLE NEWSPEED ( NEW 5 SEC VALUES)
3 VARIABLE NEWDIR
4 2VARIABLE NEWX
5 2VARIABLE NEWY
6 2VARIABLE SQUAREDFSUMS ( SUM USED AS SQUARE OF SUMS IN STD DEV)
7 2VARIABLE SUMOFSQUARES ( SUM OF SQUARED DELTA'S )
8 2VARIABLE PEAKQTAILS ( TAIL POINTERS FOR 24 HOUR PEAK WIND )
9 2VARIABLE TAILS ( TAIL POINTERS FOR QUEUES)
10 2VARIABLE SAMPCNT ( COUNT SAMPLES FOR PEAK WIND ROUTINE)
11 2VARIABLE PASSCNT ( COUNT OF VALID DATA PROCESSING CYCLES )
12 2VARIABLE FAILCNT ( COUNT OF INVALID DATA PROCESSING CYCLES)
13 2VARIABLE VALID ( FLAG USED FOR RECOVERY)
14 VARIABLE PEAKCNT & ALLOT ( VALID # OF 60 MIN & 24 HR PASSES )
15

```

## 345 LIST

```

0 ( WIND DATA VARIABLES CONTINUED                                4/8/86 )
1
2 2VARIABLE 60VALID ( 24 HR PEAK VALID FLAG)
3 2VARIABLE 24VALID ( 60 MIN PEAK VALID FLAG)
4
5 ( DATA BUFFERS )
6
7 VARIABLE TEMPBUF 30 ALLOT ( TEMPORARY HOLD BUFFER)
8 VARIABLE PRINTBUF 30 ALLOT ( PRINTER BUFFER)
9 VARIABLE DISPLAYBUF 30 ALLOT ( DISPLAY BUFFER)
10 VARIABLE AWDSBUFFER 126 ALLOT ( AWDS OUTPUT BUFFERS)
11
12 : AWDSBUF ( n -- a AWDS BUFFER FOR SENSOR n)
13     32 * AWDSBUFFER + ;
14 .S
15

```

## 346 LIST

```

0 ( WIND DATA CONSTANTS                                3/12/86)
1
2 ( THESE PERIOD CONSTANTS ARE BASED ON 5 SEC SAMPLES)
3 12 CONSTANT 1MIN
4 24 CONSTANT 2MIN
5 120 CONSTANT 10MIN
6 2 CONSTANT CELL ( INDEX USED FOR 2 BYTE INCREMENTS)
7
8 ( LIMIT CONSTANTS USED FOR SETTING INVALID DATA FLAGS)
9 12 CONSTANT 1MINLIMIT
10 24 CONSTANT 2MINLIMIT
11 120 CONSTANT 10MINLIMIT
12 720 CONSTANT 60MINLIMIT
13 17280 CONSTANT 24HRLIMIT
14
15 .S

```

## 347 LIST

```

0 ( WIND DATA - DATABASE DATA STRUCTURES                3/11/86)
1
2 ( GIVEN THE SENSOR NUMBER 'n' RETURN AN ADDRESS)
3 ( THE FOLLOWING WORDS DEFINE STORAGE BLOCKS WITH IN A
4   SENSOR FILE )
5 : SPEEDQ ( n -- a) FILESIZE * DATABASE + ;
6 : ANGLEQ ( n -- a) FILESIZE * DATABASE + DIRQ + ;
7 : GUSTQ ( n -- a) FILESIZE * DATABASE + GUSTQUE + ;
8 : INBUFFER ( n -- a) FILESIZE * DATABASE + INBUF + ;
9 : TIMEQ ( n -- a) FILESIZE * DATABASE + TIMEQUE + ;
10
11 ( DEFINITIONS FOR DATA FIELDS WITHIN THE ARRAYS ARE GIVEN
12   ON THE NEXT FEW BLOCKS)
13
14 .S
15

```

## 348 LIST

```

0 ( WIND DATA - DATABASE CONT'D                               3/12/86)
1
2 : SCRATCHPAD ( n -- a ) FILESIZE * DATABASE + SCRATCH + ;
3
4 ( DEFINITION OF CELLS WITHIN THE SCRATCHPAD ARRAY)
5 ( GIVEN A SENSOR # 'n', RETURN AN ADDRESS)
6 : XSUM2 ( n -- a ) SCRATCHPAD ; ( 2 MIN SUM OF X DATA)
7 : YSUM2 ( n -- a ) SCRATCHPAD 4 + ; ( 2 MIN SUM OF Y DATA)
8 : XSUM10 ( n -- a ) SCRATCHPAD 8 + ; ( 10 MIN SUM OF X DATA)
9 : YSUM10 ( n -- a ) SCRATCHPAD 12 + ; ( 10 MIN SUM OF Y DATA)
10 : THETA2 ( n -- a ) SCRATCHPAD 16 + ; ( 2 MIN AVE DIRECTION)
11 : THETA10 ( n -- a ) SCRATCHPAD 18 + ; ( 10 MIN AVE DIRECTION)
12 : PVHBUF ( n -- a ) SCRATCHPAD 20 + ; ( 60 MIN PEAK WIND DATA)
13 : SPEEDCK ( n -- a ) SCRATCHPAD 26 + ; ( SPEED CHECKSUM)
14 : ANGLECK ( n -- a ) SCRATCHPAD 30 + ; ( DIRECTION CHECKSUM)
15 : SUMCK ( n -- a ) SCRATCHPAD 34 + ; ( X-Y CHECKSUM) .S

```

## 349 LIST

```

0 ( WIND DATA - DATABASE CONT'D                               3/12/86)
1
2 ( QUEUS FOR 24 HOUR PEAK WIND DATA)
3 ( GIVEN n RETURN ADDRESS OF DESIRED 24 HOUR PEAK WIND QUE)
4 : PVDSPD ( n -- a ) FILESIZE * DATABASE + PVDSPD + ;
5 : PVDANGLE ( n -- a ) FILESIZE * DATABASE + PVDDIR + ;
6 : PVDTIME ( n -- a ) FILESIZE * DATABASE + PVDTIM + ;
7
8
9
10
11
12
13 .S
14
15

```

## 350 LIST

```

0 ( WIND DATA - DATABASE CONT'D                               3/12/86)
1
2 ( DEFINITION OF CELLS WITHIN THE TEMPORARY RESULT ARRAY)
3 ( CALLING THE NAME RETURNS AN ADDRESS)
4
5
6 : DIRECTION ( -- a ) TEMPBUF ;
7 : SPEED ( -- a ) TEMPBUF 2+ ;
8 : GUST ( -- a ) TEMPBUF 4 + ;
9 : DIRVAR ( -- a ) TEMPBUF 6 + ;
10 : GUSTSPRD ( -- a ) TEMPBUF 10 + ;
11 : PEAK10 ( -- a ) TEMPBUF 12 + ;
12
13 .S
14
15

```

## 351 LIST

```

0 ( WIND DATA - DATABASE CONT'D
1
2 ( DEFINITION OF CELLS WITHIN THE RESULT ARRAY)
3 : RESULTBUF ( n -- a ) FILESIZE * DATABASE + RESULTBUFFER + ;
4
5 ( THE FIRST 7 CELLS ARE THE SAME AS TEMPBUF ARRAY)
6 : PEAKRESULTS ( n -- a ) FILESIZE * DATABASE + PEAKRESULTBUF + ;
7
8 ( LAST 3 CELLS OF THE RESULT BUFFER FOR SENSOR n )
9 : PEAK60 ( n -- a ) PEAKRESULTS ;
10 : PEAK24 ( n -- a ) PEAKRESULTS 6 + ;
11 : STDDEVOUT ( n -- a ) PEAKRESULTS 12 + ;
12
13 .S
14
15

```

## 352 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1
2 : NUM-BER ( a-1 -- d, CONVERTS ASCII I/P AT a TO d, WILL NOT
3
4
5
6
7
8
9
10
11 CODE OUTPUT ( n a ) 2 POP 0 POP (2) OUT NEXT
12
13 : ?.#SENSORS ( -- n , RETURNS # OF SENSORS DEFINED - 1 )
14
15 .S

```

## 353 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1
2 : (CR) 13 EMIT 10 EMIT ;
3 : (PAGE) 12 EMIT ;
4
5 : HEX 16 BASE ! ;
6 : DECIMAL 10 BASE ! ;
7
8 CODE DIVERR
9 0 # 0 MOV IRET
10
11 : 2+! ( d, a -- ADD d TO VALUE AT ADDRESS A, DOUBLE LENGTH)
12
13
14
15

```

## 354 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1
2 : SET ( a n -- a+n a ) OVER + SWAP ;
3
4 : .0 ( n -- , PAD OUTPUT WITH n ZEROS )
5     DUP 0) IF 0 DO 0 DIGIT EMIT LOOP ELSE DROP THEN ;
6
7 : O.R ( u, n -- , SAME AS U.R EXCEPT PADS WITH ZEROS )
8     SWAP 32767 AND 0 ( # # S # ) ROT OVER - .0 TYPE ;
9
10 : SET.MSB ( n -- SET MSB OF n )
11     32768 OR ;
12
13 : O/P.TIME ( n -- OUTPUTS TIME IN hh:mm FORMAT )
14     0 ( # # # 58 HOLD # # # ) TYPE ; ( O/P TIME )
15 .S

```

## 355 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1
2 : ACT.SENSOR? ( -- n, RETURNS ACTIVE SENSOR # )
3     SENSOR# C@ 3 AND ;
4
5 CODE no.op NOP NEXT ( BREAK PT. USED FOR DEBUGGING )
6
7 .S
8
9
10
11
12
13
14
15

```

## 356 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1 HEX
2 CODE set.status ( m, a -- use mask to set bit in status )
3     W POP 1 POP ( test contents addressed by W )
4     W ) 1 TEST 0= ( is status bit clear ? )
5     IF 1 W ) OR ( set status bit )
6     1 A W ) OR ( set acknowledge bit )
7     THEN NEXT
8
9 CODE clear.status ( m, a -- use mask to clear status bits )
10    W POP 1 POP ( "a" is address of the status )
11    A W ) 1 TEST 0= ( if acknowledge bit is clear )
12    IF FFFF # 1 XOR ( form mask )
13    1 W ) AND ( clear flag in status word )
14    THEN NEXT
15 .S

```

## 357 LIST

```

0 ( INDICATOR/RECORDER SYSTEM ROUTINES )
1 HEX
2 CREATE POWER.2 ( POWER OF 2 MASK TABLE )
3   1 C, 2 C, 4 C, 8 C,
4


---


5 CODE flag.settest ( a, n -- f test flag at a using power2 n )
6   W POP POWER.2 W) 0 MOV B ( get n & setup mask)
7   W POP W ) 0 TEST B 0= NOT ( is flag set ?)
8   IF OF #B 0 XOR ( form clearing mask)
9   0 W ) AND B ( clear bit)
10  ELSE 0 0 SUB
11  THEN 0 0 PUSH NEXT ( return flag)
12 .S
13
14
15

```

## 358 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7


---


8
9
10
11
12
13
14
15

```

## 359 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 360 LIST

```

0 ( ICM7170 RTC HANDLER -- clock),)clock,?INT.CLOCK,CLKINT.STR)
1
2 HEX
3 CODE clock.latch clock IN NEXT ( READ 100DRTHS SEC )
4
5 CODE clock) ( reg# -- n, READS IN RT CLOCK REGISTER # CONTENTS)
6     2 POP clock # 2 ADD 0 # 0 MOV (2) IN 0 PUSH NEXT
7
8 CODE )clock ( n reg# -- STORES n TO REGISTER # OF RT CLOCK )
9     2 POP clock # 2 ADD 0 POP (2) OUT NEXT
10
11 CODE )clock.ram ( n, a -- place n at address a)
12     2 POP 0 POP (2) OUT NEXT
13
14 .S
15

```

## 361 LIST

```

0 ( ICM7170 RTC HANDLER -- BCD-DECI,BCD.CONV,DECIM-BCD )
1
2 : BCD.CONV ( n -- n, PERFORMS BCD CONVERSION )
3     10 /MOD 16 * + ;
4
5 : HR.SEC ( s, h, m -- d#, RETURNS TIME IN SECONDS )
6     60 * SWAP 3600 M* ROT M+ ROT M+ ;
7
8 : @DATE.TIME ( a -- RETURNS RAM TIMESTAMP y, m, d OR s, h, m )
9     DUP >R clock) I 1+ clock)
10    R) 2+ clock) ROT ROT ;
11
12
13 .S
14
15

```

## 362 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER -- GET.TIME,GET.DATE )
1
2 : GET.TIME ( -- d, RETURNS HR, MIN, & SEC. AS DOUBLE LENGTH # )
3     clock.latch
4     HOUR clock) 10000 U*
5     MINUTE clock) 100 *
6     SEC clock) + M+ ;
7
8 : GET.DATE ( -- d, RETURNS MON, DAY, & YR. AS DOUBLE LENGTH # )
9     clock.latch
10    MONTH clock) 10000 U*
11    DAY clock) 100 *
12    YEAR clock) + M+ ;
13
14 .S
15

```

## 363 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER -- !TIME, !DATE )
1
2 : .DATE ( d -- DISPLAYS DATE IN mm/dd/yy FORMAT )
3   ( # # # 47 HOLD # # 47 HOLD # # # ) TYPE ;
4
5
6 .S
7
8
9
10
11
12
13
14
15

```

## 364 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER -- !TIME, !DATE )
1
2 : GET.STAMP ( -- y, d, m, s, m, h )
3   RAMYEAR clock)
4   RAMDAY clock)
5   RAMMONTH clock)
6   RAMSEC clock)
7   RAMMIN clock)
8   RAMHOUR clock) ;
9
10 : GET.HR/MIN ( -- n, RETURNS HOUR & MINUTE )
11   clock.latch
12   HOUR clock) 100 *
13   MINUTE clock) + ;
14 .S
15

```

## 365 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER -- DISPLAY.CLOCK 8/27/86)
1
2 : GET.CLOCK ( -- y, d, m, s, m, h
3   READ TIME/DATE AS PREP FOR DISPLAY)
4   clock.latch
5   YEAR clock) DAY clock)
6   MONTH clock) SEC clock)
7   MINUTE clock) HOUR clock) PAUSE ;
8
9 : TIME.STAMP ( PLACE CURRENT TIME IN CLOCK RAM )
10   GET.CLOCK ( READ PRESENT TIME)
11   clock RAMHOUR + 6 SET ( SETUP DO LOOP)
12   DO I )clock.ram PAUSE
13   LOOP ;
14 .S
15

```

## 366 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER    MOD ROUTINES  2/2/87)
1
2 : Y2D ( year -- n, # of days since 12/31/86)
3   1987 - DAY 4 * /MOD SWAP 3 - LEAP 0! ;
4
5 : M2JUL ( month -- n, Julian day for 1st of month)
6   1- 2* D/M + @ DUP 58 ) LEAP C@ AND + ;
7
8 .S
9
10
11
12
13
14
15

```

## 367 LIST

```

0 ( ICM7170 REAL TIME CLOCK HANDLER    MOD ROUTINES  2/2/87)
1
2 : MOD.DAY ( -- n, return day # since 12/31/86 for current date)
3   clock.latch YEAR clock) DUP 87 ( ( YEAR ( 1987 ? )
4   IF 100 +
5   THEN 1900 + Y2D DAY clock)
6   MONTH clock) M2JUL + + ;
7
8 : MOD.ANYDAY ( y, m, d -- n, return day # since 12/31/86 date)
9   2>R DUP 87 ( ( YEAR ( 1987 ? )
10  IF 100 +
11  THEN 1900 + Y2D 2R) SWAP M2JUL + + ;
12
13
14 .S
15

```

## 368 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 369 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 370 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 HEX
3 : DATA. EST? ( -- n, CKS TO SEE IF DATABASES ARE REBUILDING )
4     0 SENSTAT ?.#SENSORS 1+ 2* SET ( SET UP LOOP)
5     DO I @ 8800 AND ( MASK SHORT & LONG RECOVER. BITS)
6         OR 2 ( COMBINE RECOVERY RESULTS )
7     +LOOP DUP ( ANY REBUILDING ? )
8     IF DROP 4 ( FLAG RECOVERY )
9     THEN PAUSE ;
10
11
12 .S
13
14
15

```

## 371 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 HEX
3 : COMM.ERR? ( -- n, CHK TO SEE IF ANY COMM ERR FLAGS ARE SET)
4     0 SENSTAT ?.#SENSORS 1+ 2* SET ( SET UP LOOP)
5     DO I @ 88 AND ( MASK NO RESP. & CRC-16 BITS)
6         OR 2 ( COMBINE SENSORS RESULTS )
7     +LOOP DUP ( ANY COMM. ERROR ? )
8     IF DROP 2 ( FLAG COMM. ERROR )
9     THEN PAUSE ;
10
11
12 .S
13
14
15

```

## 372 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 HEX
3 : SEN.ERROR? ( -- n, CKS IF THERE ARE ANY SENSOR ERRORS )
4 Q SENSTAT ?.#SENSORS I+ ?* SET ( SETUP SENSOR STATS)
5 DO I @ 7777 AND ( MASK SENSOR ERROR BITS )
6 OR 2 ( COMBINE SENSORS ERRORS )
7 +LOOP DUP ( ANY SENSOR PROBLEMS ?)
8 IF DROP 20 ( FLAG SENSOR ERROR )
9 THEN PAUSE ;
10
11
12 .S
13
14
15

```

## 373 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 HEX
3 CREATE GSTATMASK.TAB
4 ( MASK FOR CONVERTING I/R STATUS TO STATBYTE)
5 7000 , ( ANY PRINTER ERROR )
6 060 , ( CARRIER & 5 SEC TIMEOUT ERRORS )
7 117 , ( CPU, RAM, ROM, AWDS, OR IA COMM ERR)
8 0400 , ( LOSS OF MASTER ERROR )
9
10 CREATE SMASK.TAB ( MASK FOR SETTING BITS IN STATUS BYTE)
11 01 C, ( PRINTER ERROR )
12 02 C, ( INTER-ASSY COMM. ERROR )
13 10 C, ( MICROBD ERROR )
14 40 C, ( LOSS OF MASTER )
15 .S

```

## 374 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 CODE test.status ( tm, sm -- use test mask on status & set
3 bit in status byte if they match )
4 2 POP 1 POP
5 I/RSTATUS 1 TEST 0= NOT ( ERROR PRESENT ? )
6 IF 2 STATBYTE OR B ( FLAG GENERAL STATUS )
7 THEN NEXT
8
9 .S
10
11
12
13
14
15

```

## 375 LIST

```

0 ( INDICATOR/RECORDER STATUS CODE DISPLAY ROUTINES 1/15/87)
1
2 HEX
3 * STAT.BYTE ( -- n SCAN STATUS WORDS & FORM GENERAL STATUS)
4 0 STATBYTE C! 4 0 ( CLEAR STATUS TEMP )
5 DO GSTATMASK.TAB I 2* + @ ( GET TEST MASK)
6 I SMASK.TAB + C@ ( GET SET BIT )
7 test.status PAUSE ( CK FOR ANY ERROR )
8 LOOP STATBYTE C@ ( GET GENERAL STATUS )
9 SEN.ERROR? OR ( ADD SENSOR ERROR CODE )
10 COMM.ERR? OR ( ADD I/A COMM ERR CODE )
11 DATA.EST? OR ( ADD RECOVERY CODE )
12 DUP STATBYTE C! ; ( SAVE NEW GEN. STAT )
13
14 .S
15

```

## 376 LIST

```

0 ( McCLELLAN IND/REC DISPLAY DRIVERS 8/22/86)
1
2 HEX
3 CODE display.b ( n, -- output n as 2 nibbles to side b)
4 ( B side is D0 - D3)
5 0 POP fpi/data OUT ( display low nibble)
6 04 # 1 MOV 0 SHR V ( align upper nibble)
7 fpi/data OUT
8 NEXT
9
10
11
12
13 .S
14
15

```

## 377 LIST

```

0 ( McCLELLAN IND/REC DISPLAY DRIVERS 4/22/88)
1
2 HEX
3 CODE display.a ( n -- output n as 3 nibbles to side A)
4 ( A side is D4 - D7 )
5 0 POP 04 # 1 MOV 0 SHL V ( shift lower nibble up)
6 fpi/data OUT ( display first nibble)
7 0 SHR V fpi/data OUT ( retrieve 2nd nibble & display)
8 0 SHR V fpi/data OUT ( display 3rd nibble)
9 NEXT
10
11 CODE display.act ( n -- output single nibble)
12 0 POP 04 # 1 MOV 0 SHL V fpi/data OUT
13 OCC # 0 MOV fpi/csr OUT NEXT ( set blanking code )
14 .S
15

```

## 378 LIST

```

0 ( McCLELLAN IND/REC  FPI SETUP ROUTINE  8/22/86)
1
2 HEX
3 CODE setup.display ( side, addr -- )
4 0 POP 90 # 0 OR ( setup display RAM address)
5 fpi/csr OUT ( auto increment )
6 0 POP fpi/csr OUT
7 NEXT
8
9 : HEX.DEC ( h -- d, CONVERTS HEX TO DEC, RANGE 0-3E7 [999] )
10   OA /MOD SWAP )R ( SAVE UNITS)
11   OA /MOD 100 * ( FIND HUNDREDS)
12   SWAP 10 * + R) + ; ( FIND TENS & RECOMBINE)
13
14 .S
15

```

## 379 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY BUFFER ROUTINES  8/22/86)
1
2 : CONVERT.DISPLAYBUF ( -- CONVERTS CONTENTS OF BUFFER
3   FROM HEX TO DECIMAL, PRESERVING THE SIGN)
4   DISPLAYBUF 32 SET ( SETUP DO LOOP)
5   DO I DUP @ DUP 32767 AND ( FETCH & CONVERT VALUE)
6   HEX.DEC SWAP 0( ( TEST FOR VALID DATA)
7   IF SET.MSB ( SET MSB )
8   THEN
9   SWAP ! PAUSE 2 ( STORE VALUE)
10  +LOOP ;
11
12 .S
13
14
15

```

## 380 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY TASK  8/22/86)
1
2 : SHUFFLE.DISPLAYBUF ( -- ORGANIZE BUFFER TO MATCH DISPLAY )
3   DISPLAYBUF 4 + 6 SET
4   DO I @ PAUSE 2 ( FETCH GUST, DV1, DV2 )
5   +LOOP SWAP DISPLAYBUF 4 + 6 SET
6   DO I ! PAUSE 2 ( STORE AS DV1, DV2, GUST)
7   +LOOP ;
8
9
10 .S
11
12
13
14
15

```

## 381 LIST

```

0 ( McCLELLAN IND/REC  TIME/DATA MOD TABLES  8/22/86)
1
2 : @.FLASH ( -- RETURN FLASH FLAG) CLOCKFLAG C@ 128 AND ;
3
4 : ?FLASH ( n --- n on b IF FLAG & n(0, BLANK IT )
5     @.FLASH ( FLASH SET ? )
6     IF DUP 0( ( IS VALUE ( 0? )
7     IF DROP BLANKIT
8     THEN
9     THEN 32767 AND ; ( MASK ANY INVALID FLAG BIT )
10
11
12 .S
13
14
15

```

## 382 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY TASK 8/22/86)
1
2 : GET.STAT.GS ( -- st, gs FETCHES & CONVERTS STATUS & GUST SPD)
3     STAT.BYTE ( FETCH GENERAL STATUS BYTE)
4     I/RACK @ ( GET IND/REC ACKNOW WORD )
5     SENACK ?.#SENSORS 1+ 2* SET ( TEST ALL ACKNOW. WORDS )
6     DO I @ OR 2
7     +LOOP PAUSE ( ANY ERRORS NOT ACKNOW. ? )
8     IF SET.MSB ?FLASH ( SETUP TO FLASH STATUS CODE)
9     THEN DISPLAYBUF 10 + @ ; ( GUST SPREAD )
10
11 : DISPLAY.ACTIVE ( -- n FETCH & DISPLAY ACTIVE SENSOR # )
12     ACT.SENSOR? 1+ display.act ;
13
14 .S
15

```

## 383 LIST

```

0 ( McCLELLAN IND/REC DEVEL  TIME/DATA MOD TABLES  8/22/86)
1
2 CREATE MODTABLE ( MODULUS # FOR HR, MIN, SEC, DAY, MON, & YR )
3     24 C, 60 C, 60 C, 13 C, 32 C, 100 C,
4
5 CREATE DAYTABLE ( MODULUS # FOR # OF DAYS IN EACH MONTH )
6     32 C, 29 C, 32 C, 31 C, 32 C, 31 C,
7     32 C, 32 C, 31 C, 32 C, 31 C, 32 C,
8
9 : CONF.CODE ( -- n, RETURNS DISPLAY CODE FOR IND/REC CONFIG. )
10     CONFIG C@ 12 AND ?DUP ( MASTER OR REGULAR ? )
11     IF 8 = ( MASTER ? )
12     IF 1 ( MASTER CODE ) ELSE 3 ( REGULAR CODE ) THEN
13     ELSE 2 ( BACKUP CODE )
14     THEN 4080 OR ; ( BLANK LEFT 2 DIGITS )
15 .S

```

## 384 LIST

```

0 ( McCLELLAN IND/REC DEVEL  TIME/DATA MOD TABLES  8/22/86)
1
2 : FIX.DATE ( n, MOD # -- n, SETS TO PROPER DATE )
3   OVER 1 ( ( n ( 1 2 ) )
4   IF 1- SWAP DROP ( SET TO MOD # - 1 )
5   ELSE MOD DUP 0= ( MOD RESULT = 0 ? )
6     IF 1+
7     THEN
8     THEN ;
9
10
11
12
13
14
15

```

## 385 LIST

```

0 ( McCLELLAN IND/REC DEVEL  TIME/DATA MOD TABLES  8/22/86)
1
2 : DAY.FIX ( n, FIELD# -- n, SETS TO PROPER DAY )
3   DROP MONTH clock) DUP 1- ( GET RTC MONTH )
4   DAYTABLE + C@ SWAP 2 = ( MONTH = FEB ? )
5   IF YEAR clock) 4 MOD 0 = ( LEAP YEAR ? )
6   IF 1+ ( ACCT FOR LEAP DAY )
7   THEN
8   THEN FIX.DATE ;
9
10 : SET.RTC ( n, MOD # -- n, SETS TO PROPER TIME OR YEAR )
11   OVER 0 ( ( NEGATIVE n ? )
12   IF 1- SWAP DROP ( SET TO MOD # - 1 )
13   ELSE MOD
14   THEN ;
15

```

## 386 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY TASK  CLOCK.SET ROUTINE  8/27/86)
1
2 : CLOCK.SET ( CLOCKFLAG, n, U/DFLAG -- CLOCKFLAG, n  SET CLOCK)
3   16 AND ( DEC. FIELD DOWN ? )
4   IF 1-
5   ELSE 1+ ( INC. FIELD UP)
6   THEN CLOCKFLAG DUP C@
7     199 AND SWAP C! ( RESET FLAG)
8     OVER 7 AND DUP 4 = ( DAY FIELD ? )
9     IF DAY.FIX ( SET DAY )
10    ELSE DUP MODTABLE + C@ ( GET OTHER MOD. #)
11    SWAP 3 = ( MONTH FIELD ? )
12    IF FIX.DATE ( SET MONTH )
13    ELSE SET.RTC ( SET TIME/YEAR )
14    THEN
15    THEN PAUSE ; .S

```

## 387 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY TASK  CLOCK.SET ROUTINE  8/27/86)
1
2 : ?CLOCK.SET ( n, CLOCKFLAG -- n,  SET CLOCK/DATA )
3           SWAP OVER 48 AND ?DUP ( UP OR DOWN SET ?)
4           IF CLOCK.SET
5             DUP ROT 7 AND
6             1+ )clock           ( UPDATE CLOCK REGISTER )
7           ELSE SWAP DROP
8           THEN ;
9
10 : CLOCK.FLASH ( n, CLOCKFLAG -- n,  SET n & FLASH n )
11           ?CLOCK.SET @.FLASH ( GET FLASH FLAG)
12           IF DROP BLANKIT ( FLASH n)
13           THEN ;           ( SET n)
14 .S
15

```

## 388 LIST

```

0 ( McCLELLAN IND/REC  DIPLAY TASK  8/22/86)
1
2 : CLOCK.DISP ( WRITES CLOCK DATA TO B-SIDE OF 8279 KBDI )
3   GET.CLOCK ( GET RTC REGISTERS DATA)
4   BSIDE 0 setup.display 6 0 ( SETUP DISPLAY INTF)
5   DO CLOCKFLAG C@ DUP 64 AND ( RUN/SET FLAG SET ?)
6   IF DUP 7 AND I = ( SET THIS FIELD ?)
7   IF CLOCK.FLASH ( FLASH OR SET IT )
8   ELSE DROP
9   THEN
10  ELSE DROP
11  THEN DUP BLANKIT = NOT ( NOT BLANK CHAR ? )
12  IF BCD.CONV ( CONVERT TO BCD # )
13  THEN display.b PAUSE ( DISPLAY CLOCK FIELD )
14  LOOP ;
15 .S

```

## 389 LIST

```

0 ( McCLELLAN IND/REC  DISPLAY TASK  8/27/86)
1
2 : NORM.DISP ( DISPLAY DATA ON NORMAL DATA DISPLAY )
3   DISPLAYBUF 10 SET ASIDE 0 setup.display
4   DO I @ ?.FLASH ( READ VALUE, TEST FLASH)
5   display.a PAUSE 2
6   +LOOP
7   DISPLAY.ACTIVE ( DISPLAY ACTIVE SENSOR)
8   CLOCK.DISP ( DISPLAY REAL TIME CLOCK )
9   GET.STAT.GS ?.FLASH ( OUTPUT GUST SPREAD)
10  display.b display.b ; ( OUTPUT STATUS)
11 .S
12
13
14
15

```

## 390 LIST

```

0 ( McCLELLAN IND/REC DISPLAY TASK ROUTINE 8/27/86)
1
2 HEX
3 CREATE SWMASK EQ C, DO C, BO C, ZO C,
4
5 CODE led.hdlr ( n -- ) 0 POP mcs/reg OUT NEXT
6
7 : LED.ON ( n -- TURNS SWITCH LED n ON )
8     SWMASK + C@ LEDSTATE C@ OF AND OR DUP
9     LEDSTATE C! led.hdlr ;
10
11 : LED.OFF ( -- TURN SWITCH LEDS OFF )
12     LEDSTATE C@ FO OR DUP LEDSTATE C! led.hdlr ;
13
14 : DIS.PEAK.TIME ( n -- CONVERTS TO BCD & DISPLAYS TIME n )
15     OA /MOD 10 * + display.b ;

```

## 391 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE PEAK WIND DISPLAY 9/16/86)
1
2 : PEAK.OUT 2 0 DO ?.FLASH display.a PAUSE LOOP ;
3
4 : PEAK.DISP ( n # -- DISPLAYS n PEAK WIND DATA AT DISPLAYBUF+#)
5     ASIDE 0 setup.display ( DISPLAY A SIDE)
6     DISPLAYBUF + >R ( SAVE DISPLAYBUF ADDR. )
7     I 2+ @ I @ ( GET SPEED & DIR)
8     PEAK.OUT 4095 display.a ( DISPLAY & BLANK DV WINDOW)
9     DISPLAYBUF 30 + @ PEAK.OUT ( DISPLAY STD DEV & PEAKTYP)
10    DISPLAY.ACTIVE BSIDE 0 setup.display
11    R) 4 + @ 256 /MOD ( DISPLAY TIME OF OCCURANCE)
12    DIS.PEAK.TIME display.b 5 0
13    DO 255 display.b PAUSE ( BLANK SEC, DATE & GS )
14    LOOP GET.STAT.GS DROP display.b ; ( DISPLAY STATUS )
15 .S

```

## 392 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 : PEAK10.DISP ( RECORDER 10 MIN PEAK WIND DISPLAY MODE )
3     16 ( 10 HEX) 12 PEAK.DISP ;
4
5 : PEAK60.DISP ( RECORDER HOURLY PEAK WIND DISPLAY MODE )
6     96 ( 60 HEX) 18 PEAK.DISP ;
7
8 : PEAK24.DISP ( RECORDER DAILY PEAK WIND DISPLAY MODE )
9     36 ( 24 HEX) 24 PEAK.DISP ;
10
11
12
13 .S
14
15

```

## 393 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 : CASE.DISP ( n -- DISPLAYS RECORDER DATA FOR MODE n )
3     DUP 1- LED.ON   ( TURN SW LED ON )
4     CASE
5         1 OF NORM.DISP ENDOF
6         2 OF PEAK10.DISP ENDOF
7         3 OF PEAK60.DISP ENDOF
8         4 OF PEAK24.DISP ENDOF
9         ENDCASE ;
10
11
12
13 .S
14
15

```

## 394 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 HEX
3 CODE parse.byte ( n -- c1, c2 generate status mode codes)
4     ( take word & form code with middle nibble as a blank)
5     0 POP 7777 # 0 AND
6     0 HI 2 MOV B 4 # 1 MOV 2 SHL V      ( shift nibble up)
7     F00 # 2 AND 0 HI 1 MOV B OF # 1 AND ( save 4th & 3rd nib)
8     2 1 OR F0 # 1 OR 1 PUSH            ( return n4.f.n3 )
9     0 2 MOV 4 # 1 MOV 2 SHL V          ( repeat for lower byte )
10    F00 # 2 AND OF # 0 AND             ( save 1st & 2nd nibble )
11    2 0 OR F0 # 0 OR 0 PUSH           ( return n2.f.n1)
12    NEXT
13
14 .S
15

```

## 395 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 HEX
3 CODE test.pattern ( n -- display n in all locations )
4     A0 #B 0 MOV fpi/csr OUT ( setup display ram)
5     50 #B 0 MOV fpi/csr OUT ( set to auto inc)
6     16 # 1 MOV 0 POP
7     BEGIN fpi/data OUT      ( write to all locations)
8     LOOP NEXT
9
10 : ALARM.ON F7 LEDSTATE C@ AND DUP LEDSTATE C! led.hdlr ;
11
12 : ALARM.OFF 08 LEDSTATE C@ OR DUP LEDSTATE C! led.hdlr ;
13
14 CREATE TEST.PAT 99 C, 88 C, 77 C, 66 C, 55 C, 44 C, 33 C, 22 C,
15     11 C, 00 C, .S

```

## 396 LIST

```

0 ( STATUS DISPLAY MODE ROUTINE )
1
2 : LITE.SHOW ( n -- DISPLAY n ON COUNT DOWN DISPLAY TEST )
3     DUP TEST.PAT + @           ( FETCH TEST PATTERN )
4     test.pattern         ( DISPLAY n )
5     DUP 3 AND LED.ON 9 =       ( STEP THRU LEDS )
6     IF ALARM.ON                ( SOUND ALARM LAST LOOP )
7     THEN
8     BEGIN PAUSE
9         REFRESHFLAG C@         ( WAIT FOR 0.5 SEC )
10    UNTIL 0 REFRESHFLAG C! ;   ( CLEAR FLAG )
11
12
13
14
15

```

## 397 LIST

```

0 ( STATUS DISPLAY MODE ROUTINE )
1
2 : TEST.DISP ( TEST DISPLAY FOR THE STATUS DISPLAY MODE )
3     BEGIN 10 0
4         DO I LITE.SHOW         ( DISPLAY # )
5             CLOCKFLAG C@       ( GET SET/RUN STATUS )
6             64 AND 0=          ( RUN MODE ? )
7         IF LEAVE
8             THEN PAUSE
9             LOOP ALARM.OFF
10            CLOCKFLAG C@       ( GET SET/RUN STATUS )
11            64 AND 0= PAUSE    ( RUN MODE ? )
12    UNTIL ;
13
14
15 .S

```

## 398 LIST

```

0 ( STATUS DISPLAY MODE ROUTINE )
1
2 HEX
3 : COMM.DISP ( # -- c, FORMS CODE FOR SENSOR COMM. STAT # )
4     8088 AND 0                 ( MASK DESIRED BITS )
5     OVER 80 AND                ( MULTIPLE CRC-16 ERROR ? )
6     IF 4 OR                    ( FLAG CRC-16 ERROR )
7     THEN OVER 8 AND            ( NO RESP. ERROR ? )
8     IF 2 OR                    ( FLAG NO RESP. ERROR )
9     THEN SWAP 0(               ( LONG TERM RECOVERY ? )
10    IF 1 OR                    ( FLAG LONG RECOVERY )
11    THEN FO OR ;               ( ADD BLANK CODE )
12
13
14 .S
15

```

## 399 LIST

```

0 ( STATUS DISPLAY MODE ROUTINE )
1
2 : STAT.DISP2 ( # cc sc1 sc2 c1 c2 -- DISP STATUS CODES )
3 ASIDE 0 setup.display 5 0 ( SETUP SIDE A)
4 DO display.a PAUSE ( DISPLAY STATUS CODES )
5 LOOP DISPLAY.ACTIVE ( DISPLAY ACTIVE SENSOR )
6 BSIDE 0 setup.display 6 0 ( SETUP SIDE B)
7 DO 32768 ?.FLASH display.b PAUSE ( FLASH 0'S IN CLOCK)
8 LOOP COMM.DISP display.b ( DISP ANY COMM ERROR)
9 GET.STAT.GS DROP
10 display.b ; ( DISPLAY GENERAL STATUS)
11
12 .S
13
14
15

```

## 400 LIST

```

0 ( STATUS DISPLAY MODE ROUTINE )
1
2 : STAT.DISPLAY ( -- DISPLAY I/R & SENSOR STATUS WORDS)
3 CLOCKFLAG C@ 64 AND ( TEST DISPLAY ? )
4 IF TEST.DISP
5 ELSE CONF.CODE ( FORM CONFIGURATION CODE )
6 SENSOR# C@ 16 / ( GET SELECTED SENSOR # )
7 1- DUP LED.ON ( TURN ON SELECTED SENSOR LED)
8 2* SENSTAT + @ ( POINT TO SENSOR STAT)
9 SWAP OVER
10 parse.byte PAUSE ( FORM SENSOR CODE )
11 I/RSTATUS @ ( FETCH IND/REC STATUS)
12 parse.byte ( FORM IND/REC CODE )
13 STAT.DISP2 ( DISPLAY STATUS CODES )
14 THEN ;
15 .S

```

## 401 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 : REFRESH.DISP ( REFRESHES FRONT PANEL DATA DISPLAY )
3 STATDISFLAG C@ ( IN STATUS DISP. MODE ? )
4 IF STAT.DISPLAY
5 ELSE CONFIG C@ 16 AND ( INDICATOR ASSY ? )
6 IF NORM.DISP
7 ELSE SENSOR# C@ 16 / ( GET REC. DISPLAY MODE )
8 CASE.DISP ( DISPLAY DATA )
9 THEN
10 THEN CLOCKFLAG C@ 128 XOR
11 CLOCKFLAG C! ( RESET FLASH BIT )
12 TIME.STAMP ; ( SAVE CLOCK REGISTERS )
13
14 .S
15

```

## 402 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 : NEW.DISPLAY ( PREPARES DISPLAY DATA FOR NEW 5 SEC UPDATE )
3 0 DISFLAG C! STATDISFLAG C@ 0= ( NOT IN STAT DISP ? )
4 IF SENSOR# C@ CONFIG C@ 16 AND ( INDICATOR ASSY ? )
5 IF DUP 16 / ?DUP ( ACT. OR MOMENT. SEN DISPLAY ? )
6 IF SWAP DROP 1- ( MOMENTARY SENSOR DISPLAY )
7 DUP LED.ON ( TURN SW. LED ON )
8 ELSE LED.OFF ( TURN SW. LED OFF )
9 THEN
10 ELSE 7 AND ( REC. ASSY ACTIVE SENSOR DISPLAY )
11 THEN RESULTBUF DISPLAYBUF ( COPY SENSOR DATA )
12 32 CMOVE PAUSE ( TO DISPLAY TEMP. BUFFER )
13 CONVERT.DISPLAYBUF SHUFFLE.DISPLAYBUF
14 THEN ; .S
15

```

## 403 LIST

```

0 ( FRONT PANEL DISPLAY TASK ROUTINE )
1
2 : DISP.DATA ( FRONT PANEL DISPLAY TASK HANDLER )
3 10 0
4 DO I LITE.SHOW PAUSE ( PWR-UP COUNT DOWN DISPLAY)
5 LOOP ALARM.OFF
6 BEGIN REFRESHFLAG C@ ( TIME FOR 500 mS REFRESH ? )
7 IF 0 REFRESHFLAG C! ( CLEAR FLAG )
8 DISPFLAG C@ ( TIME FOR 5 SEC UPDATE ? )
9 IF NEW.DISPLAY ( COPY DATA FROM DATABASE )
10 THEN REFRESH.DISP ( REFRESH DISPLAY )
11 THEN PAUSE ( CHANGE ASSY SIMULATOR ? )
12 AGAIN ;
13
14 .S
15

```

## 404 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 417 LIST

```

0 ( PRINTER PAGING AND HEADER ROUTINE
1
2 : PRINT.HEADER ( HEADER FOR RECORDER )
3
4 ." 10 MIN. PEAK 50 MIN. PEAK 24 HOUR PEAK" CR
5 ." DATE TIME DIR SPD GST DIR - VAR GSP "
6 3 0
7 DO ." DIR SPD TIME "
8 LOOP ." SD AS ST " CR ;
9
10 : HEADER.TIME? ( -- , PRINT HEADER WHEN HEADERFLAG IS SET)
11     HEADERFLAG C@ DUP 29 = ( TIME FOR NEW HEADER ? )
12     IF CR PRINT.HEADER NOT
13     ELSE 1+
14     THEN HEADERFLAG C! ;
15 .S

```

## 418 LIST

```

0 ( ROUTINES TO PRINT THE TIME AND DATE
1
2 : PRINT.DATE ( PRINT DATE OF LINE PRINTOUT IN mm/dd/yy FORMAT )
3     GET.DATE .DATE 3 SPACES ;
4
5 : PRINT.TIME ( PRINT TIME OF LINE PRINTOUT )
6     GET.HR/MIN O/P.TIME 3 SPACES ;
7
8 : PRINT.PKTIME ( n -- PRINTOUT PEAKWIND TIME OF OCCURRENCE )
9     O/P.TIME 5 SPACES ;
10
11 : STAT.UNACK? ( -- f, CKS IF ANY SYS ERR ARE UNACKNOWLEDGED )
12     I/RACK @
13     SENACK ?.#SENSORS 1+ 2* SET ( GET IND/REC ACKNOW WORD )
14     DO I @ OR PAUSE 2 ( TEST ALL ACKNOW. WORDS )
15     +LOOP ;

```

## 419 LIST

```

0 ( PRINTER ROUTINE
1
2 : PRT.PK.WIND ( PRINTS 10 MIN, HRLY, & DAILY PEAK WIND DATA )
3     PRINTBUF 12 + 3 0 ( SETUP PTR FOR PEAK WINDS DATA )
4     DO DUP /DATA 3 0.R 2 *SPACES ( PEAK DIR. )
5     2+ DUP /DATA 3 0.R 2 *SPACES ( PEAK SPD )
6     2+ DUP @ PRINT.PKTIME 2+ PAUSE ( TIME OF OCCUR. )
7     LOOP /DATA 3 0.R 2 *SPACES ; ( PRINT STD DEV )
8
9 : PRT.SYS.DATA ( n -- PRINTS OUT SYSTEM DATA )
10     1+ 2 0.R 2 SPACES ( PRINT ACTIVE SENSOR # )
11     STATBYTE C@ HEX 2 0.R DECIMAL ( PRINT SYS STATUS )
12     STAT.UNACK? ( ANY UNACKNOWLEDGE ERR ? )
13     IF 42 EMIT ( PRINT STAR AFTER STATUS )
14     THEN 13 EMIT 96 SPACES CR ;
15 .S

```

## 420 LIST

```

0 ( PRINTER ROUTINE                                     4/5/86)
1
2 : PRINT.OUT ( PRINTS OUT ACTIVE SENSOR DATA ON PRINTER )
3   15 EMIT ( OKI DATA 17.1 PRINTER CPI COMMAND)
4   HEADER.TIME? ( CHECK IF TO PRINT HEADER )
5   PRINT.DATE PRINT.TIME ( PRINT OUT DATE & TIME )
6   SENSOR# C@ 3 AND DUP ( GET ACTIVE SENSOR # )
7   RESULTBUF PRINTBUF ( COPY ACT. SENSOR DATA )
8   32 CMOVE PAUSE ( TO PRINTER TEMP BUFFER)
9   PRINTBUF 12 SET
10  DO I /DATA 3 O.R ( PRINT 1ST 6 DATA VALUES)
11    3 *SPACES 2 ( FLAG ANY INVALID DATA )
12  +LOOP 2 SPACES
13    PRT.PK.WIND ( PRINT OUT PEAK WIND DATA )
14    PRT.SYS.DATA ; ( PRINT OUT SYSTEM DATA )
15 .S

```

## 421 LIST

```

0 ( PRINTER TASK ROUTINE )
1
2 : PRINTER ( TASK TO PRINT OUT ACT. SENSOR DATA EVERY 1 MINUTE )
3   29 HEADERFLAG C! ( FLAG TO PRINT OUT HEADER )
4   PRIME.PRINTER 3000 0 ( DELAY AFTER START-UP PRIME)
5   DO PAUSE LOOP ( GET PRINTER STATUS )
6   BEGIN PRINTER.STATUS? 0= ( TIME TO PRINT OUT ? )
7   PRINTFLAG C@ ( CLEAR FLAG, STATUS OKAY ?)
8   IF 0 PRINTFLAG C!
9     IF PRINT.OUT
10    THEN
11    ELSE DROP
12    THEN PAUSE
13    AGAIN ;
14 .S
15

```

## 422 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

423 LIST

```
0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

---

---

424 LIST

```
0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

---

425 LIST

```
0 ( McCLELLAN IND/REC  AWDS I/O ROUTINES          9/24/86)
1
2 : {TYPE} ( AWDS VECTOR TYPE ROUTINE )
3     PTR @ C@ 1 PTR +!      ( GET FIRST CHARACTER )
4     CTR @ 1- CTR !
5     awds/data OUTPUT STOP ;    ( O/P LAST CHAR )
6
7 CODE o/p.enable? ( -- f, CKS IF CS & DM ARE ACTIVE THRU CTS I/P)
8     awds/csr IN 32 # 0 AND    ( MASK CTS BIT )
9     0 PUSH NEXT
10
11 : SEND.TIME ( PRINT TIME OF LINE PRINTOUT )
12     GET.HR/MIN O/P.TIME ;
13
14 .S
15
```

## 426 LIST

```

0 ( MESSAGE COUNTER FOR AWDS                                4/16/86)
1
2 : MESSAGE.COUNT ( -- n , RETURN VALUE OF COUNTER MODULO 100)
3 MESSAGE# DUP C@ 1+ ( FETCH, INCREMENT)
4 100 MOD DUP ROT C! ;
5
6 : SEND.DATE ( PRINT DATE OF LINE PRINTOUT IN mm/dd/yy FORMAT )
7   GET.DATE .DATE ;
8
9 : VALID.DATA ( a -- ? , RETURN ASCII '?' IF DATA IS INVALID)
10   48 SWAP 32 SET      ( SETUP DO LOOP)
11   DO I @ 0(          ( TEST FOR NEGATIVE VALUES)
12   IF DROP 63 LEAVE   ( LEAVE '?' ON STACK IF 0( )
13   THEN PAUSE 2
14   +LOOP ;
15 .S

```

## 427 LIST

```

0 ( AWDS XMITTER TEST ROUTINE                                4/16/86 )
1
2 : TEST.UARTB ( TESTS AWDS TRANSMITTER LOOPBACK )
3   256 I/RSTATUS      ( SETUP STATUS )
4   CHARBUFFB C@      ( GET 1ST CHAR LOOPBACKED)
5   70 =              ( IS IT 'F' CHAR ? )
6   IF clear.status   ( FLAG NO XMIT ERR )
7     0 UARTBERR C!   ( CLEAR ERROR CTR )
8   ELSE UARTBERR C@ 1+ ( INC. ERROR CTR )
9     2 MIN DUP       ( NO MORE THAN 3 )
10    UARTBERR C! 2 = ( ERRORS = 2 ? )
11    IF set.status    ( FLAG XMIT ERROR )
12    ELSE clear.status ( FLAG NO XMIT ERR )
13    THEN
14    THEN 0 CHARBUFFB C! ;
15 .S

```

## 428 LIST

```

0 ( AWDS OUTPUT ROUTINE                                    4/16/86)
1
2 : NON/ACT.O/P ( n -- O/P/S ACTIVE/NONACTIVE CODE & SYSTEM STATUS)
3   SENSOR# C@ 15 AND OVER = ( ACTIVE SENSOR ? )
4   IF 65                    ( O/P 'A' )
5   ELSE 78                  ( O/P 'N' )
6   THEN EMIT SPACE STATBYTE C@
7   HEX 2 O.R DECIMAL CR    ( O/P SYSTEM STATUS )
8   0=                      ( FIRST SENSOR ? )
9   IF TEST.UARTB PAUSE     ( AWDS XMITTER TEST )
10  THEN ;
11
12
13 .S
14
15

```



## 429 LIST

```

0 ( AWDS OUTPUT ROUTINE                                     4/16/86)
1
2 : AWDS.OUT ( n -- , OUTPUT DATA IN AWDS BUFFER n )
3   SEND.DATE SPACE SEND.TIME SPACE ( OUTPUT DATE & TIME)
4   DUP AWDSBUF 12 SET ( SETUP DO LOOP)
5   DO I @ 3 0.R SPACE PAUSE 2 ( OUTPUT DATA AND A SPACE)
6   +LOOP DUP AWDSBUF 12 + 3 0 ( SET LOOP FOR PEAK DATA )
7   DO DUP @ 3 0.R SPACE
8     2+ DUP @ 3 0.R SPACE
9     2+ DUP @ 0/P.TIME SPACE 2+ PAUSE
10  LOOP @ 3 0.R SPACE ( OUTPUT STD DEV)
11      NON/ACT.0/P ; ( ACT CODE & STATUS )
12
13 .S
14
15

```

## 430 LIST

```

0 ( AWDS MESSAGE LEADER                                     4/12/89)
1
2 : MESSAGE.HEADER ( n -- ,AWDS MESSAGE LEADER FOR SNESOR n)
3   DUP 0= ( FIRST SENSOR ? )
4   IF 1 UARBFLG C! ( GET 1ST LOOPBACK CHAR )
5   THEN ." FMQ13V" ( SYSTEM NOMENCLATURE)
6   DUP 1+ 2 0.R ( SENSOR #)
7   MESSAGE.COUNT 2 0.R ( MODULO 100 MESSAGE CTR)
8   AWDSBUF VALID.DATA EMIT ( PRINT '?' OR '0')
9   90 3 0.R SPACE ; ( FIXED DATA COUNT)
10
11
12 .S
13
14
15

```

## 431 LIST

```

0 ( AWDS TASK ROUTINE                                     5/5/86)
1
2 : AWDS ( -- OUTPUT DATA FOR ALL CONFIGURED SENSORS)
3   BEGIN AWDSFLAG C@ ( TIME FOR OUTPUT ? )
4   IF 0 AWDSFLAG C! ( CLEAR FLAG)
5   o/p.enable? ( 'CS' & 'DM' ACTIVE ? )
6   IF ?.#SENSORS 1+ DUP 0 ( READ # OF SENSORS)
7   DO I RESULTBUF ( MOVE DATA TO )
8   I AWDSBUF 32 CMOVE PAUSE ( AWDS O/P BUFFER )
9   LOOP 0
10  DO I MESSAGE.HEADER ( START AWDS OUTPUT)
11  I AWDS.OUT PAUSE ( OUTPUT DATA)
12  LOOP
13  THEN
14  THEN PAUSE
15  AGAIN ; .S

```

432 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

433 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

434 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

435 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

436 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

437 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

438 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

439 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

440 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

## 441 LIST

```

0 ( HEAD & TAIL ROUTINES FOR QUEUES                               5/30/86)
1
2 ( THESE ROUTINES KEEP TRACK OF THE QUEUE TAIL & HEAD POINTERS )
3
4 : TAIL ( n -- cnt, RETURNS OFFSET TO QUEUE TAIL FOR SENSOR n)
5     TAILS + C@ ;
6
7 : 2.MIN ( a, n -- a, RETURNS POINTER TO 2 MINUTE HEAD)
8     TAIL 192 + 240 MOD + ;
9
10 : 10.MIN ( a, n -- a, RETURN POINTER TO 10 MINUTE HEAD)
11     TAIL 240 MOD + ;
12
13 : UPDATE.Q ( n -- INCREMENTS TAIL POINTER OF SENSOR n MOD 240)
14     TAILS + DUP C@ 2+ 240 MOD SWAP C! ;
15 .S

```

## 442 LIST

```

0 ( 2 OR 10 ? - TEST CONFIGURATION BYTE                           4/16/86)
1
2 : 2.OR.10 ( -- f , TEST CONFIGURATION BYTE RETURN TRUE
3     FOR 2 MINUTES)
4     CONFIG C@ DUP 32 AND ( COPY & MASK OFF 2/10 BIT )
5     SWAP 16 AND OR ; ( MASK I/R BIT , OR THE RESULTS )
6
7
8
9
10
11
12
13
14
15 .S

```

## 443 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 444 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 445 LIST

```

0 ( Q* QUAD LENGTH MULITPLICATION) HEX
1 CODE Q* ( d d -- q )
2 83 C, EC C, 08 C, 8B C, FC C, 8B C, 45 C, 0E C, 8B C, 4D C,
3 0A C, F7 C, E1 C, 89 C, 55 C, 04 C, 89 C, 45 C, 06 C, 31 C,
4 C0 C, 89 C, 45 C, 02 C, 89 C, 05 C, 8B C, 45 C, 0C C, F7 C,
5 E1 C, 01 C, 45 C, 04 C, 11 C, 55 C, 02 C, 83 C, 15 C, 00 C,
6 8B C, 4D C, 08 C, 8B C, 45 C, 0E C, F7 C, E1 C, 01 C, 45 C,
7 04 C, 11 C, 55 C, 02 C, 83 C, 15 C, 00 C, 8B C, 45 C, 0C C,
8 F7 C, E1 C, 01 C, 45 C, 02 C, 11 C, 15 C, FA C, 83 C, C4 C,
9 10 C, FF C, 75 C, 06 C, FF C, 75 C, 04 C, FF C, 75 C, 02 C,
10 FF C, 35 C, FB C, NEXT
11
12 CODE Q+ ( q q - q )
13 8B C, FC C, 8B C, 45 C, 06 C, 01 C, 45 C, 0E C, 8B C, 45 C,
14 04 C, 11 C, 45 C, 0C C, 8B C, 45 C, 02 C, 11 C, 45 C, 0A C,
15 8B C, 05 C, 11 C, 45 C, 08 C, 83 C, C4 C, 08 C, NEXT .S

```

## 446 LIST

```

0 ( 14-bit Fixed-point fraction arithmetic)
1 ( From Poly-FORTH level 3 )
2
3 16384 CONSTANT +1
4
5 CODE *. ( n f - n) 0 POP 1 POP 1 IMUL
6 0 SHL 2 RCL 0 SHL 2 RCL 2 PUSH NEXT
7
8 CODE /. ( n n - f) 1 POP 2 POP 0 0 SUB
9 2 SAR 0 RCR 2 SAR 0 RCR 1 IDIV 0 PUSH NEXT
10
11 CODE U/ ( u u - u ) W POP 0 POP 2 2 SUB W DIV 0 PUSH NEXT
12
13 .S
14
15

```

## 447 LIST

```

0 ( Trigonometry for 14-bit fractions)
1 ( From Poly-FORTH level 3 )
2
3 CODE TRIANGLE ( a - f)  0 POP  0 SHL  0 SHL
4   0< IF  0 NEG  THEN  +1 # 0 SUB  0 NEG  0 PUSH  NEXT
5
6 CODE 90- ( a - a)  0 POP  4096 # 0 SUB  0 PUSH  NEXT
7
8   ( Hart 3300:  1.57079  -.64589  .07943  -.00433)
9 : COS ( a - f)  TRIANGLE DUP DUP *.  DUP -71 *.
10   1301 + OVER *.  -10582 + *.  9352 + OVER *.  + PAUSE ;
11
12 : SIN ( a - f)  90- COS ;
13
14 .S
15

```

## 448 LIST

```

0 ( Arc-tangent)
1 ( From Poly-FORTH level 3 )
2
3 CODE !SET  0 POP  0 ROR  CS NOT IF
4   LODS  THEN  R ) RCL  NEXT
5
6 ( Hart 4960:  .15920  -.05270  .02680  .41421)
7 HERE HEX  -0C00 ,  0400 ,  3400 ,  -3C00 ,
8   1400 ,  -1C00 ,  -2C00 ,  2400 ,  DECIMAL
9
10 : ATAN ( n n - a)  DUP 0< >R ABS
11   SWAP DUP 0< !SET NEGATE 2DUP > !SET SWAP /.
12   DUP -6768 + SWAP 6786 *.  +1 + /. PAUSE
13   DUP DUP *.  DUP 438 *.  -864 + *.  2607 + *.
14   R) 2* LITERAL + @ + ABS PAUSE ;
15 .S

```

## 449 LIST

```

0 ( Angle conversion & Square Root )
1 ( From Poly-FORTH level 3 )
2
3 : DEG ( n - f)  360 /. ;
4
5 : REV ( f - n)  0 18000 8192 M*/ DROP ;
6
7 CODE SQRT ( d - n, RETURNS SQUARE ROOT FOR DOUBLE-LENGTH # )
8   2 POP  0 POP  I PUSH  W W SUB  W I MOV  16 # 1 MOV
9   BEGIN  0 SHL  2 RCL  I RCL  0 SHL  2 RCL  I RCL
10   W SHL  W SHL  W INC  W I CMP  CS NOT IF
11   W I SUB  W INC  THEN  W SHR
12   LOOP  I POP  W PUSH  NEXT
13
14 .S
15

```

## 450 LIST

```

0 ( Quad-length Square Root Routine )
1
2 HEX CODE QSQRT ( q - d )
3 83 C, EC C, 04 C, 8B C, FC C, 31 C, D2 C, 31 C, C0 C, 89 C,
4 05 C, 89 C, 45 C, 02 C, B9 C, 20 C, 00 C, D1 C, 65 C, 0A C,
5 D1 C, 55 C, 08 C,
6 D1 C, 55 C, 06 C, D1 C, 55 C, 04 C, D1 C, D0 C, D1 C, D2 C,
7 D1 C, 65 C, 0A C, D1 C, 55 C, 08 C, D1 C, 55 C, 06 C, D1 C,
8 55 C, 04 C, D1 C, D0 C, D1 C, D2 C, D1 C, 25 C, D1 C, 55 C,
9 02 C, D1 C, 25 C, D1 C, 55 C, 02 C, 83 C, 05 C, 01 C, 83 C,
10 55 C, 02 C, 00 C, 3B C, 55 C, 02 C, 72 C, 12 C,
11 75 C, 04 C, 3B C, 05 C, 72 C, 0C C, 2B C, 05 C, 1B C, 55 C,
12 02 C, 83 C, 05 C, 01 C, 83 C, 55 C, 02 C, 00 C, D1 C, 6D C,
13 02 C, D1 C, 1D C, E2 C, B1 C, 8B C, 05 C, 8B C, 55 C, 02 C,
14 83 C, C4 C, 0C C, 50 C, 52 C, NEXT
15 .S

```

## 451 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 452 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 453 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 454 LIST

```

0 ( VECTORIAL AVERAGE ROUTINE                               3/4/86)
1 ( X, Y -- RESULTANT )
2 : DVEC.AVE ( 2x, 2y -- 2resultant, R=[x*x + y*y]SQRT )
3     2)R DABS 2DUP Q* 2R) PAUSE
4     DABS 2DUP Q* Q+ QSQRT PAUSE ; ( COMPUTE DOUBLE ROOT)
5
6 CODE SCALE.DOWN ( n -- n/128  ROUNDED UP )
7     0 POP 64 # 0 ADD      ( GET n & ADD 64)
8     7 # 1 MOV            ( SET LOOP )
9     BEGIN 0 SHR LOOP     ( DIVIDE BY 128 )
10    0 PUSH NEXT          ( PUSH RESULTS )
11
12 : 24RND 12288 0 D+ 24576 M/ ; ( 2*[16384 + 8192])
13 .S
14
15

```

## 455 LIST

```

0 ( McCLELLAN IND/REC   CHECKSUM ROUTINE           5/20/87)
1
2 : SUM.SUMS ( n -- d  ADD RUNNING SUM AS CHECK SUM)
3     )R 0 0 I XSUM2 2@ D+ ( ADD EACH SUM)
4     I YSUM2 2@ D+ I XSUM10 2@ D+
5     R) YSUM10 2@ D+ ;    ( LEAVE SUM ON STACK)
6
7 : UPDATE.CHKSUM ( n -- UPDATE SPEED & DIRECTION CHECKSUMS)
8     )R I SPEEDQ I 10.MIN @ ( FETCH OLDEST)
9     0 DNEGATE I SPEEDCK 2+! ( SUBTRACT FROM SUM)
10    NEWSPEED @ 0 I SPEEDCK 2+! ( ADD NEWEST )
11    I ANGLEQ I 10.MIN @      ( REPEAT FOR DIRECTION)
12    0 DNEGATE I ANGLECK 2+!
13    NEWDIR @ 0 I ANGLECK 2+!
14    I SUM.SUMS R) SUMCK 2! ; ( COMPUTE X-Y CHECK)
15 .S

```

## 456 LIST

```

0 ( RECT. TO POLAR & POLAR TO RECT. CONVERSION          9/26/86)
1
2 : REC.POLAR ( x, y -- m, a )
3     2DUP 2>R >R 2 M* R) 2 M*                ( SCALE UP)
4     DVEC.AVE 1 0 D+ 2 M/                    ( COMPUTE MAGNITUDE)
5     2R) SWAP ATAN ;                          ( COMPUTE DIRECTION)
6
7
8 : POLAR.REC ( m, a -- 2y, 2x )
9     DEG 2DUP COS M* 1 128 M*/ 2>R          ( COMPUTE X )
10    SIN M* 1 128 M*/ 2R) ;                 ( COMPUTE Y )
11
12
13 .S
14
15

```

## 457 LIST

```

0 ( SUM.YX10 - COMPUTE 10 MINUTE RUNNING SUM OF X&Y 4/9/86)
1
2 : SUM.XY10 ( n -- n SUBTRACT OLDEST & ADD NEWEST)
3     DUP >R                ( SAVE n)
4     ( COMPUTE RUNNING SUM FOR Y)
5     YSUM10 DUP 2@         ( I' IS THE SENSOR #)
6     I SPEED@ I 10.MIN @   ( FETCH OLDEST SPEED & ANGLE)
7     I ANGLE@ I 10.MIN @   POLAR.REC 2>R ( COMPUTE X & Y)
8     D- NEWY 2@ D+ ROT 2! ( SUBTRACT OLDEST, ADD NEWEST & STORE)
9
10    ( COMPUTE RUNNING SUM FOR X)
11    J XSUM10 DUP 2@ 2R) D- ( FETCH SUM & SUBTRACT OLDEST)
12    NEWX 2@ D+ ROT 2!     ( SUBTRACT OLDEST, ADD NEWEST & STORE )
13    R) ;                  ( RESTORE STACKS)
14 .S
15

```

## 458 LIST

```

0 ( SUM.YX2 - COMPUTE 2 MINUTE RUNNING SUM OF X & Y    4/9/86)
1
2 : SUM.XY2 ( n -- n SUBTRACT OLDEST & ADD NEWEST)
3     DUP >R                ( SAVE n)
4     ( COMPUTE RUNNING SUM FOR Y)
5     YSUM2 DUP 2@         ( I' IS THE SENSOR #)
6     I SPEED@ I 2.MIN @   ( FETCH OLDEST SPEED & ANGLE)
7     I ANGLE@ I 2.MIN @   POLAR.REC 2>R ( COMPUTE X & Y)
8     D- NEWY 2@ D+ ROT 2! ( SUBTRACT OLDEST, ADD NEWEST & ST)
9
10    ( COMPUTE RUNNING SUM FOR X)
11    J XSUM2 DUP 2@ 2R) D- ( FETCH SUM & SUBTRACT OLDEST)
12    NEWX 2@ D+ ROT 2!     ( SUBTRACT OLDEST, ADD NEWEST & STO)
13    R) ;                  ( RESTORE STACKS)
14 .S
15

```

## 459 LIST

```

0 ( XY - COMPUTE LATEST SUMS                                4/9/86 )
1
2 : UPDATE.SDXY ( n -- UPDATE TAIL POINTER & STORE LATEST DATA)
3   >R I UPDATE.CHKSUM                                     ( UPDATE CHECK SUMS)
4   NEWSPEED @ I SPEEDQ I TAIL + !                       ( STORE SPEED)
5   NEWDIR @ I ANGLEQ R) TAIL + ! ;                       ( STORE DIRECTION)
6
7 : SUM.XY ( n -- sumy, sumx)
8   DUP >R
9   SUM.XY10
10  2.OR.10 ( DO WE NEED 2 MINUTE SUMS?)
11  IF SUM.XY2 YSUM2 2@ I XSUM2 2@
12  ELSE YSUM10 2@ I XSUM10 2@ ( PLACE Y & X ON STACK)
13  2)R 1 5 M*/ 2R) 1 5 M*/ ( SCALE DOWN )
14  THEN R) UPDATE.SDXY ; ( UPDATE SPEED, DIR, X, & Y QUE)
15 .S

```

## 460 LIST

```

0 ( COMPUTE NEW X & Y VALUES AND SUMS                      3/12/86)
1
2 : COMPUTE.XY ( -- COMPUTE X & Y FROM NEWSPEED & NEWDIR )
3   NEWSPEED @ ( FETCH LATEST SPEED)
4   NEWDIR @ ( FETCH ANGLE & CONVERT TO RECTANGULAR)
5   POLAR.REC NEWX 2! NEWY 2! ; ( SAVE RESULTS)
6
7
8
9
10 .S
11
12
13
14
15

```

## 461 LIST

```

0 ( AVE.SPEED - VECTORIALLY AVERAGED SPEED                 3/12/86)
1
2 : AVE.SPEED ( n -- , COMPUTE THE AVERAGE SPEED FOR SENSOR n)
3   COMPUTE.XY ( COMPUTE LATEST X & Y)
4   SUM.XY
5   2)R 1 16 M*/ 2R) 1 16 M*/ ( SCALE DOWN SUMS)
6   DVEC.AVE ( COMPUTE VECTORIAL AVE, LEAVE RESULT )
7   24RND ( SCALE DOWN SPEED)
8   SPEED ! ; ( STORE RESULTS)
9
10
11 .S
12
13
14
15

```

## 462 LIST

```

0 ( GET X&Y FOR THETABAR                                10/22/86 )
1
2 : THETA.XY2 ( -- x, y GET 2 MINUTE X&Y SUMS )
3   I' XSUM2 2@ 2DUP D0= ( SUM = 0 ? )
4   IF DROP ( MAKE SINGLE LENGTH )
5   ELSE 3000 M/
6   THEN I' YSUM2 2@ 2DUP D0=
7   IF DROP
8   ELSE 3000 M/
9   THEN ; ( 2 MINUTE )
10 .S
11
12
13
14
15

```

## 463 LIST

```

0 ( THETA.XY10 - GET X&Y FOR THETABAR                    10/22/86 )
1
2 : THETA.XY10 ( -- x, y GET 10 MINUTE X&Y SUMS )
3   I' XSUM10 2@ 2DUP D0= ( SUM = 0 ? )
4   IF DROP
5   ELSE 15000 M/
6   THEN I' YSUM10 2@ 2DUP D0=
7   IF DROP
8   ELSE 15000 M/
9   THEN ; ( 10 MINUTE )
10 .S
11
12
13
14
15

```

## 464 LIST

```

0 ( COMP.THETA - WIND DIRECTION                            3/12/86 )
1
2 : COMP.THETA ( x, y -- theta, COMPUTE THE ANGLE )
3   SWAP ATAN ABS REV 50 +
4   100 U/ DUP 0= ( ANGLE = 0 ? )
5   IF DROP 360 ( ANGLE = 360 DEGREES )
6   THEN ;
7
8
9
10
11
12
13
14 .S
15

```

## 465 LIST

```

0 ( THETA.BAR - MIN AVERAGE DIRECTION                               3/12/86)
1
2 : THETA.BAR ( n -- , )
3   >R SPEED @ 0= ( SPEED = 0 ? )
4   IF 0 ( CALM WINDS HAVE 0 DIRECTION)
5   ELSE THETA.XY2 COMP.THETA I THETA2 !
6   THETA.XY10 COMP.THETA I THETA10 !
7   I 2.OR.10 ( OUTPUT 2 MINUTE VALUE ?)
8   IF THETA2 @ ( PLACE PROPER VALUE ON STACK)
9   ELSE THETA10 @ ( 10 MINUTE VALUE )
10  THEN
11  THEN DIRECTION ! R) DROP ; ( STORE RESULTS) .S
12 .S
13
14
15

```

## 466 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 467 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 468 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 469 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 470 LIST

```

0 ( WRAP ROUTINE FOR MX.V MN.V & MX.MN           4/29/88)
1
2 : WRAP? ( o, n -- o'  WRAP OFFSET BACKWARDS AROUND END OF QUE )
3           ( o IS OFFSET FROM MAX.V, ETC.)
4           ( TAIL RETRIEVES THE TAIL POINTER FOR SENSOR n)
5           TAIL SWAP - -240 MOD ;
6
7 : OFF.SET ( n -- o  RETURN OFFSET TO MAX)
8           OFFSET @ 2* SWAP WRAP? ;
9
10 : O.QUE? ( a -- f  CHECK FOR ALL ZERO QUE)
11           0 SWAP 240 SET   ( LEAVE 0 FLAG, SETUP DO LOOP )
12           DO I @ 0= NOT   ( TEST FOR 0 )
13           IF NOT LEAVE   ( SET FLAG & LEAVE)
14           THEN 2         ( INC. TO NEXT CELL)
15           +LOOP ;      .S

```

## 471 LIST

```

0 ( MAX.V - FIND MAXVALUE IN QUE & SAVE OFFSET          4/05/88)
1
2 : MAX.V ( a, i, l, n -- GIVEN Q-ADDR., INDEX, & LENGTH
3             FIND MAXIMUM & ITS OFFSET FOR SENSOR n )
4     )R 0 MXVALUE ! 0          ( INIT. MXVALUE & SETUP LOOP )
5     DO 2DUP I * J WRAP? +    ( COMPUTE ADDRESS)
6     @ SCALE.DOWN            ( FIND WHOLE KNOTS)
7     DUP MXVALUE @ )        ( COMPARE DATA & MXVALUE)
8     IF MXVALUE !           ( SAVE MAXIMUM VALUE)
9     I OFFSET !             ( SAVE MAX & OFFSET)
10    ELSE DROP              ( CLEAR STACK OF LAST VALUE)
11    THEN PAUSE
12    LOOP 2DROP R) DROP ;   ( CLR STACK OF ADDR, INDEX, n )
13
14 .S
15

```

## 472 LIST

```

0 ( MX.MN - FIND MAX & MIN IN A QUEUE          3/13/86)
1
2 ( Q-ADDR, INDEX, LENGTH -- )
3 : MX.MN ( a, i, l, n -- GIVEN ADDR, INDX & LENGTH,
4             FIND MAX & MIN FOR SENSOR n)
5     )R 0 MXVALUE ! 32000 MNVALUE ! 0    ( INIT. VALUES )
6     DO 2DUP I * J WRAP? + DUP          ( CALCULATE ADDRESS)
7     @ DUP MNVALUE @ (             ( MINIMUM VALUE ? )
8     IF MNVALUE !           ( SAVE MINIMUM )
9     ELSE DROP
10    THEN @ DUP MXVALUE @ )           ( MAXIMUM VALUE ? )
11    IF MXVALUE !           ( SAVE AS MAXIMUM)
12    ELSE DROP
13    THEN PAUSE
14    LOOP 2DROP R) DROP ;
15 .S

```

## 473 LIST

```

0 ( GUSTSPREAD          8/11/89)
1
2 : GUST.SPREAD ( n -- COMPUTE LATEST GUST SPREAD FOR SENSOR n )
3     DUP )R          ( SAVE SENSOR #)
4     SPEEDQ 2 1MIN ( SET STACK AS ADDR., INDEX, LENGTH -- )
5     R) MX.MN      ( FIND MAX & MIN)
6     MXVALUE @ MNVALUE @ -    ( COMPUTE THE GUSTSPREAD)
7     SCALE.DOWN 99 MIN    ( LIMIT TO 99 KNOTS)
8     GUSTSPRD ! ; ( PLACE IN RESULT BUFFER)
9
10 : SET.6024VALID? ( n -- SET 60 MIN & 24 HR VALID FLAGS)
11     DUP )R 2* PEAKCNT + @ DUP ( GET PEAK COUNT)
12     60MINLIMIT (             ( HAS ( 60 MIN PASSED ? )
13     IF 1 ELSE 0 THEN I 60VALID + C!
14     24HRLIMIT (             ( HAS ( 24 HRS PASSED ? )
15     IF 1 ELSE 0 THEN R) 24VALID + C! ; .S

```

## 474 LIST

```

0 ( PK.WD - SEARCH SPEED QUEUE AND FIND PEAK DATA          3/13/86 )
1
2 ( CALLED BY PEAK WIND)
3
4 : PK.WD ( ANGLEQ, TIMEQ, SPEEDQ, INDX, LENGTH, n --
5           TIME, DIR, SPEED FOR SENSOR n)
6           ( MAXV USES TOP 3 ARGUMENTS)
7           MAX.V ( FIND MAX & OFFSET)
8           I' OFF.SET DUP ( GET OFFSET AND COMPUTE INDEX)
9           ROT + @ ( GET ASSOCIATED TIME)
10          SWAP ROT + @ ( GET ASSOCIATED DIR)
11          MXVALUE @ ; ( GET SPEED )
12
13 .S
14
15

```

## 475 LIST

```

0 ( PVHR - DETERMINE HOURLY PEAK WIND DATA          3/13/86)
1
2 ( IF THE LATEST 5SEC PEAK WIND DATA IS
3   A NEW HOURLY PEAK FOR SENSOR n, STORE DATA IN PVHBUF)
4
5 : PVH.R ( TIME, DIR, SPEED, n -- TIME, DIR, SPEED)
6   >R 2>R J SWAP >R ( SAVE DATA, R-STK = DIR, SPEED, TIME --)
7   PVHBUF DUP ( COMPUTE PVH ADDRESS)
8   4 + I SWAP ! ( STORE NEW TIME)
9   DUP 2+ J SWAP ! ( STORE ASSOCIATED DIRECTION)
10  I' SWAP ! ( STORE ASSOCIATED SPEED)
11  R) 2R) R) DROP ; ( CLEAR RETURN STACK)
12
13 : SAMPCNT.@ ( n -- # RETURN COUNT & INCREMENT TO LIMIT)
14           SAMPCNT + DUP C@ DUP 1+ 120 MIN ROT C! ;
15 .S

```

## 476 LIST

```

0 ( PEAK WIND LATEST 5 SEC VALUE OF 10 MIN PEAK          3/12/86)
1
2 : PEAK.WIND ( n -- , FIND LATEST 5 SECOND VALUE OF THE
3             10 MINUTE PEAK WIND FOR SENSOR n)
4 ( SETUP STACK FOR PK.WD WITH Q-ADDRESS FOR DIRECTION,
5   TIME, & SPEED ALONG WITH INDEX, LENGTH & SENSOR #)
6
7   DUP >R ANGLEQ I TIMEQ I SPEEDQ CELL 10MIN I
8   PK.WD DUP I PVHBUF @ ( NOT ( PK.WD ) OR = TO PVH ? )
9   I SAMPCNT.@ 119 ) AND ( AND 10 MIN. PAST SINCE XX:55 ? )
10  IF I PVH.R ( STORE NEW 60 MIN PEAK WIND)
11  THEN SWAP ( REORGANIZE STACK - TIME, DIR, SPEED)
12  PEAK10 DUP ( COMPUTE ADDR. OF RESULT ARRAY)
13  >R ! I 2+ ! ( STORE DIRECTION & SPEED)
14  R) 4 + ! R) DROP ; ( STORE TIME & RETURN n )
15 .S

```

## 477 LIST

```

0 ( UPDATE 24 HOUR PEAK WIND QUEUES                               3/3/87)
1
2 : UPDATE.PVD ( n -- UPDATE QUES FOR 24 HR PEAK WIND DATA )
3   ( THESE QUES ARE 48 BYTES LONG)
4   PEAKQTAILS + DUP C@ 2+ 48 MOD SWAP C! ;
5
6 : PEAKQ.TAIL ( n-- cnt, RETURNS OFFSET TO QUE TAIL FOR SENSOR n)
7   PEAKQTAILS + C@ ;
8
9 : PEAK.WRAP? ( o, n -- o WRAP OFFSET BACKWARDS AROUND END OF QU)
10  PEAKQ.TAIL SWAP - -48 MOD ;
11
12 : PEAK.OFFSET ( n -- o RETURN OFFSET TO MAX PEAK24 )
13   OFFSET @ 2* SWAP PEAK.WRAP? ;
14 .S
15

```

## 478 LIST

```

0 ( PVHGET - RETRIEVE LATEST VALUE OF PVH FOR SENSOR n   3/12/86)
1
2 : PVH.GET ( n -- SPEED, DIRECTION, TIME )
3   ( RETRIEVE THE LATEST 60 MIN PEAK WIND DATA FROM
4     THE SCATCH PAD AREA OF SENSOR n)
5   PVHBUF ( COMPUTE ADDRESS OF PVHBUFFER)
6   DUP DUP @ 0 ROT ! ( PUT SPEED ON STACK AND ZERO PV)
7   SWAP DUP 2+ @ ( GET DIRECTION)
8   SWAP 4 + @ ; ( GET TIME)
9
10 : CLEAR.SAMPCNT ( n -- CLEAR SAMLE COUNT OF SENSOR n)
11   SAMPCNT + 0 SWAP C! ;
12 .S
13
14
15

```

## 479 LIST

```

0 ( PVHPUT - PUT PVH DATA IN RESULT & PVD QUE SENSOR n   3/12/86)
1
2 : PVH.PUT ( SPEED, DIRECTION, TIME, n-- )
3   ( PLACE 60 MIN PEAK WIND DATA IN PVH BUFFER)
4   TEMP ! ( STORE n)
5   )R 2)R ( PLACE DATA ON R-STACK)
6   TEMP @ PEAK60 ( GET OUTPUT BUFFER PVH ADDRESS)
7   DUP I SWAP ! ( STORE DIRECTION)
8   DUP 2+ I' SWAP ! ( STORE SPEED)
9   4 + J SWAP ! ( STROE TIME)
10  ( NOW PLACE DATA IN PVD QUE)
11  TEMP @ DUP PVDANGLE OVER PEAKQ.TAIL +
12  R) SWAP ! ( GET PVDANGLE Q & STORE)
13  DUP PVDSPEED OVER PEAKQ.TAIL + R) SWAP ! ( STORE SPEED)
14  DUP PVDTIME SWAP PEAKQ.TAIL + R) SWAP ! ; ( STORE TIME)
15 .S

```

## 480 LIST

```

0 ( PVDOUT - PLACE PVD DATA IN RESULT BUFFER FOR SENSOR n 3/12/86)
1
2 ( PVD STRUCTURE - DIRECTION, SPEED, TIME )
3 : PVD.OUT ( n -- , PLACE 24 HR PEAK WIND DATA IN RESULTBUF )
4   ( COMPUTE ADDRESS OF MAX PVD DATA)
5   DUP DUP TEMP ! ( SAVE n )
6   PEAK.OFFSET DUP )R ( COMPUTE INDEX & SAVE)
7   SWAP PVDTIME + @ ( FETCH TIME)
8   I TEMP @ PVDSPEED + @ ( FETCH SPEED)
9   R) TEMP @ PVDANGLE + @ ( FETCH DIRECTION)
10
11 ( SETUP FOR RESULT BUFFER )
12   TEMP @ PEAK24 ( RETURNS PVD RESULT BUFFER ADDRESS)
13   DUP )R ! ( SAVE ADDRESS AND STORE DIRECTION DATA)
14   I 2+ ! ( STORE SPEED)
15   R) 4 + ! ; ( STORE TIME) .S

```

## 481 LIST

```

0 ( MAX.P - FIND MAXVALUE IN QUE & SAVE OFFSET 3/13/86)
1
2 : MAX.P ( a, i, l, n -- GIVEN Q-ADDR., INDEX, & LENGTH
3   FIND MAXIMUM & ITS OFFSET FOR SENSOR n )
4   )R 0 MXVALUE ! 0 ( INIT MXVALUE & SETUP DO LOOP)
5   DO 2DUP I * J PEAK.WRAP? + ( COMPUTE ADDRESS)
6   @ DUP MXVALUE @ ) ( COMPARE DATA & MXVALUE)
7   IF MXVALUE ! ( SAVE VALUE)
8   I OFFSET ! ( SAVE MAX & OFFSET)
9   ELSE DROP ( CLEAR STACK OF LAST VALUE)
10  THEN PAUSE
11  LOOP 2DROP R) DROP ; ( CLR STACK OF ADD, INDEX , n)
12
13 .S
14
15

```

## 482 LIST

```

0 ( HOUR55 - FUNCTIONS PERFORMED AT 55 PASTED THE HOUR 3/12/86)
1
2 : HOUR.55 ( n -- , PERFORM HOURLY TASK AT 55 PAST)
3   55FLAG C@ ( FETCH FLAG)
4   IF DUP 55FLAG SWAP flag.settest ( TEST FLAG)
5   IF DUP )R PVH.GET ( PLACE LATEST PVH IN PVD QUES )
6   I PVH.PUT ( STORE PVH VALUE IN BUFFER & IN PVD QUEUE)
7   I PVDSPEED 2 24 I ( SET UP FOR MAXV)
8   MAX.P ( MAX.P FINDS OFFSET TO PEAK & SAVES IN "OFFSET" )
9   I PVD.OUT ( PVDOUT MOVES PVD DATA FORM QUE TO OUTBUF)
10  I UPDATE.PVD ( UPDATE TAIL POINTERS)
11  I CLEAR.SAMPcnt ( CLEAR SAMPLE COUNTER)
12  R) SET.6024VALID? ( SET 60 MIN OR 24 HR FLAGS)
13  ELSE DROP
14  THEN
15  ELSE DROP THEN ; .S

```

## 483 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 484 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 485 LIST

```

0 ( GUST2.TEST - 5 SEC GUST USNIG 2 MIN ALGORITHM          3/14/86)
1
2 : GUST2.TEST ( GUSTSPREAD, PV, SBAR -- GUST)
3   ( DETERMINE GUST AS GIVEN BELOW USING 5 SEC SAMPLE)
4   ( IF [SBAR > 0] & [PV - SBAR >= 5] & [GS >= 10];
5     THEN GUST = PV, ELSE GUST = 0 )
6
7   2DUP ROT 0 ) ( GS, PV, PV ,SBAR, SBAR -- SBAR)0 ?)
8
9     ( YES, [(PV - SBAR NOT < 5) & (GS NOT < 10)]NOT ?)
10  IF - 5 < NOT ROT 10 < NOT AND NOT
11    IF DROP 0 ( DROP PV & ENTER 0)
12    THEN
13  ELSE 2DROP 2DROP 0 ( SBAR NOT > 0, SO GUST = 0)
14  THEN ; ( STACK HAS GUST VALUE ON IT)
15 .S

```

## 486 LIST

```

0 ( GUST10.TEST - 5 SEC GUST USING 10 MIN ALGORITHM   3/14/86)
1
2 : GUST10.TEST ( PV, SBAR -- GUST , 10 MIN GUST ALGORITHM)
3   ( DETERMINE GUST AS GIVEN BELOW USING 5 SEC SAMPLE)
4   ( IF [SBAR > 0] & [PV - SBAR >= 5];
5     THEN GUST = PV, ELSE GUST = 0 )
6
7   2DUP ROT 0 )      ( PV, PV , SBAR, SBAR -- SBAR ) 0 ?)
8   IF - 5 (          ( YES, [(PV - SBAR NOT( 5)] ?)
9     IF DROP 0      ( DROP PV & ENTER 0)
10    THEN
11  ELSE 2DROP DROP 0 ( SBAR NOT) 0, SO GUST = 0)
12  THEN ;           ( STACK NOW HAS GUST VALUE ON IT)
13 .S
14
15

```

## 487 LIST

```

0 ( GUST.NEW SETUP STACK FOR GUST2.TEST OR GUST10.TEST   4/29/88)
1
2 ( PLACE RESULTS IN GUST QUEUE FOR PROCESSING BY "GUST.GUSTY")
3
4 : GUST.NEW ( n -- n , SETUP STACK FOR 2 OR 10 MIN ALGORITHM)
5   )R ( SAVE n)
6   PEAK10 2+ @ SPEED @      ( PV, SPEED -- )
7   2.OR.10                  ( 2 MIN ?)
8   IF GUSTSPRD @           ( 2 MIN, SO GET GUSTSPREAD)
9     ROT ROT GUST2.TEST    ( SETUP STACK & CALL GUST2.TEST)
10  ELSE GUST10.TEST        ( 10 MIN, CALL GUST10.TEST)
11  THEN I GUSTQ
12    I TAIL + ! R) ;      ( STORE RESULTS IN GUST QUE)
13 .S
14
15

```

## 488 LIST

```

0 ( GUST.FIND - FIND LATEST 5 SEC GUST                   4/10/86 )
1
2 : GUST.GUSTY ( n -- , FIND LATEST GUST VALUE FOR SENSOR n)
3   GUST.NEW ( COMPUTES GUST USING 2 OR 10 MIN ALGORITHM)
4   GUSTQ 0.QUE?          ( SETUP STACK FOR 0.QUE?)
5   IF PEAK10 2+ @       ( NOT =0, FETCH 10 MIN PEAK )
6   ELSE 0                ( GUST QUE ALL 0'S, 0 GUST )
7   THEN GUST ! ;        ( PLACE ANSWER IN RESULTS ARRAY)
8
9 .S
10
11
12
13
14
15

```

## 489 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 490 LIST

```

0 ( DIRECTION VARIABILITY                                1/6/87)
1
2 CODE ).180? ( #, n -- # IF n ) +-180 ; +-360 to #)
3     0 POP 1 POP ( GET THETA)
4     180 # 0 CMP 0) ( n ) 180 ?)
5     IF 360 # 1 SUB ( n = n - 360 )
6     ELSE -180 # 0 CMP 0( ( n < -180 ? )
7     IF 360 # 1 ADD ( n = n + 360 )
8     THEN
9     THEN 1 PUSH NEXT
10
11 .S
12
13
14
15

```

## 491 LIST

```

0 ( DIRECTION VARIABILITY                                1/6/87)
1
2 : FIND.DV ( -- DETERMINE DIRECTION VARIABILITY VALUES)
3     THETA0 @ DUP MNVALUE @ + DUP 0( ( DV1 = THETA0 + MIN)
4     IF 360 + ( IF DV1 < 0 , ADD 360 DEGREES)
5     THEN ?DUP NOT ( RESULTS = 0 ?)
6     IF 360 ( SET TO 360)
7     THEN DIRVAR ! ( STORE IN DV1 POSITION OF TEMPBUF)
8     MNVALUE @ + DUP 360 ) ( DV2 = THETA0 + MAX, ) 360 ?)
9     IF 360 - ( IF YES, SUBTRACT 360 )
10    THEN ?DUP NOT ( RESULTS = 0 ?)
11    IF 360
12    THEN DIRVAR 2+ ! ; ( STORE IN DV2 POSITON )
13
14 .S
15

```

## 492 LIST

```

0 ( DIRECTION VARIABILITY                                1/6/87)
1
2 : ).360? ( -- f, IS MAX - MIN ) 360 ? SETUP TEST FLAG)
3     MXVALUE @ MNVALUE @ - DUP ( COMPUTE DIFFERENCE )
4     360 = SWAP 360 > OR ; ( ) OR = TO 360 ?)
5
6 : MAX.MIN ( n -- DETERMINE IF n IS NEW MAX OR MIN)
7     DUP MNVALUE @ ( TEST MIN VALUE)
8     IF MNVALUE ! ( SAVE NEW MINIMUM)
9     ELSE DUP MXVALUE @ ) ( TEST FOR NEW MAXIMUM)
10    IF MXVALUE ! ( SAVE IF NEW MAXIMUM)
11    ELSE DROP ( ELSE DROP n)
12    THEN
13    THEN ;
14
15 .S

```

## 493 LIST

```

0 ( DIRECTION VARIABILITY                                1/6/87)
1
2 : FIND.EXTREMES ( a,i,l,n -- n SEARCH THRU QUE FOR EXTREMES)
3     )R 0 MXVALUE ! 0 MNVALUE ! 1 ( SAVE n & SET VARS)
4     DO 2DUP I * J WRAP? + @ ?DUP ( FETCH THETA FROM QUE)
5     IF THETA0 @ - DUP ( SKIP IF =0, COMPUTE DELTA-THETA )
6     ).180? DUP ( CORRECT IF OVER +- 180 )
7     DELTADelta @ - ( COMPUTE DELTA-DELTA-THETA )
8     ).180? ( CORRECT DELTA-THETA IF OVER +- 180)
9     DUP DELTADelta ! ( SAVE AS NEW DELTA-THETA )
10    MAX.MIN ).360? ( FIND MAX/MIN & TEST IF ) 360 )
11    IF DIRECTION @ DUP DIRVAR 2! ( SET DV = AVE. DIR.)
12    12345 DELTADelta ! LEAVE ( USE DELTA-DELTA AS FLAG)
13    THEN
14    THEN PAUSE
15    LOOP 2DROP R) DROP ; ( DROP a,i & n ) .S

```

## 494 LIST

```

0 ( DIRECTION VARIABILITY                                1/6/87)
1
2 : DIR.VAR ( n -- FIND DIRECTION VARIABILITY FOR SENSOR n)
3     )R 0 THETA0 ! 0 DELTADelta ! ( SAVE n, INIT. VARIABLE)
4     I ANGLEQ 2 10MIN 0 ( GET THETA0 )
5     DO 2DUP I * J WRAP? + @ ?DUP
6     IF THETA0 ! LEAVE ( SAVE FIRST NON-0 THETA )
7     THEN
8     LOOP 2DROP THETA0 @ ( THETA0 NOT= 0?)
9     IF I ANGLEQ 2 10MIN I ( SETUP FOR SEARCH)
10    FIND.EXTREMES DELTADelta @ 12345 = NOT ( 360 LOOP?)
11    IF FIND.DV ( NO, SO FIND DIRECTION VARIABILITY)
12    THEN
13    ELSE 0 DIRVAR ! 0 DIRVAR 2+ ! ( ALL THETA,S = 0)
14    THEN R) UPDATE.Q ; .S ( UPDATE QUE POINTER )
15 .S

```

495 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

496 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

497 LIST

0 ( STANDARD DEVIATION OF DIRECTION  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

12/8/86)

## 498 LIST

```

0 ( STANDARD DEVIATION OF DIRECTION                                12/8/86)
1
2 : SUM.DELTAS ( n -- n COMPUTE SUM OF SQUARES & SUM OF DELTAS )
3   DUP THETA10 @ ( FETCH 10 MIN AVE. DIRECTION)
4   OVER ANGLE@ 240 SET ( GET ADDRESS AND SET UP LOOP)
5   DO DUP I @ ?DUP ( SKIP IF = 0 )
6   IF - ( DUP THETA10 & FETCH THETAi )
7   DUP ).180? ABS ( COMPUTE min | delta | )
8   DUP 0 SQUAREOFSUMS 2+! ( ADD TO SUM)
9   DUP M* SUMOFSQUARES 2+! ( SQUARE & ADD TO SUM )
10  1 #THETAS +! ( INC # OF THETAS)
11  ELSE DROP ( DROP EXTRA THETA10)
12  THEN PAUSE 2 ( INC TO NEXT CELL )
13  +LOOP DROP ;
14 .S
15

```

## 499 LIST

```

0 ( STANDARD DEVIATION OF DIRECTION                                12/8/86)
1
2 : FIND.STDDEV ( -- std.dev )
3   SUMOFSQUARES 2@ 10000 ( SCALE BY 10000)
4   #THETAS @ M*/ ( DIV SUM OF SQUS BY N)
5   SQUAREOFSUMS 2@
6   DABS DROP DUP M* ( SQUARE SUMS )
7   10000 #THETAS @ DUP * ( COMPUTE N*N)
8   M*/ D- DABS SQRT PAUSE ( COMPUTE STANDARD DEVIATION )
9   50 + 100 / ; ( ROUND & SCALE DOWN)
10
11
12
13
14
15

```

## 500 LIST

```

0 ( STANDARD DEVIATION OF DIRECTION                                12/8/86)
1
2 : STD.DEV ( n -- )
3   STDFLAG C@
4   IF DUP STDFLAG SWAP flag.settest
5   IF 0 0 SQUAREOFSUMS 2!
6   0 0 SUMOFSQUARES 2! 0 #THETAS ! ( INIT. VARIABLES)
7   SUM.DELTAS ( COMPUTE DELTAS & SUMS)
8   FIND.STDDEV ( COMPUTE STD. DEV. )
9   SWAP STDDEVOUT ! ( STORE IN TEMPBUF)
10  ELSE DROP
11  THEN
12  ELSE DROP ( DROP n)
13  THEN ; .S
14
15

```

## 501 LIST

```

0 ( ROUTINE TO SET PEAK WIND DATA AS INVALID          5/15/86)
1
2 : BAD.PEAK ( lcnt, a -- a, SET MSB OF PEAK WIND SPEED & DIR.)
3     SWAP 0
4     DO 3 0
5     DO 2+ I 2 = NOT          ( NOT TIME OF OCCURRENCE?)
6     IF DUP @ SET.MSB OVER !
7     THEN PAUSE
8     LOOP
9     LOOP ;
10
11 .S
12
13
14
15

```

## 502 LIST

```

0 ( McCLELLAN SET.INVALID ROUTINES                    1/14/87)
1
2 : SET.BAD ( a -- FETCH DATA FROM a, SET MSB & STORE)
3     DUP @ SET.MSB SWAP ! ;
4
5 : SET.2GROUP ( SET SPEED & DIRECTION INVALID)
6     SPEED SET.BAD DIRECTION SET.BAD ;
7
8 : SET.GS ( SET GUST SPREAD INVALID)
9     GUSTSPRD SET.BAD ;
10
11 .S
12
13
14
15

```

## 503 LIST

```

0 ( McCLELLAN SET.INVALID ROUTINES                    1/14/87)
1
2 : SET.24PEAK ( SET 24 HOUR PEAK WIND DATA INVALID )
3     1 TEMPBUF 22 + BAD.PEAK DROP ;
4
5 : SET.60PEAK ( SET 60 MIN PEAK WIND DATA INVALID )
6     1 TEMPBUF 16 + BAD.PEAK DROP ;
7
8 : SET.10GROUP ( SET 10 MINUTE DATA INVALID )
9     GUST SET.BAD          ( SET GUST & DIR. VARIABILITY INVALID)
10    DIRVAR DUP SET.BAD 2+ SET.BAD
11    TEMPBUF 30 + SET.BAD          ( SET STD. DEV. INVALID)
12    1 TEMPBUF 10 + BAD.PEAK ( SET 10 MIN PEAK WIND INVALID)
13    DROP ;
14 .S
15

```

## 504 LIST

```

0 ( 1st.PROCESS      INITILIALIZE DATABASE ON 1ST POWER-UP 2/11/87)
1
2 : SET.INVALID ( n f -- SET DATABASE AS INVALID)
3     IF SET.24PEAK SET.60PEAK
4     THEN SET.26GROUP SET.6S SET.10GROUP
5         TEMPBUF SWAP RESULTBUF 32 CMOVE PAUSE ;
6
7 : SET.SUM2 ( dy dx n -- INITIALIZES SENSOR n XSUM2 & YSUM2 )
8     DUP >R XSUM2 2! R) YSUM2 2! ;
9
10 : SET.SUM10 ( dy dx n -- INITIALIZES SENSOR n XSUM10 & YSUM10 )
11     DUP >R XSUM10 2! R) YSUM10 2! ;
12
13
14 .S
15

```

## 505 LIST

```

0 ( 1st.PROCESS      INITILIALIZE DATABASE ON 1ST POWER-UP 2/11/87)
1
2 : INIT.SUMS ( n -- INITIALIZE RUNNING SUMS )
3     >R COMPUTE.XY
4     NEWX 2@ NEWY 2@ 2.OR.10
5     IF 24
6     ELSE 120
7     THEN DUP >R 1 M*/ 2SWAP
8         R) 1 M*/ R) 2.OR.10
9     IF SET.SUM2
10    ELSE SET.SUM10
11    THEN ;
12 .S
13
14
15

```

## 506 LIST

```

0 ( 1st.PROCESS      INITILIALIZE DATABASE ON 1ST POWER-UP 2/11/87)
1
2 : INTQ ( #, a -- PLACE # IN EACH WORD OF QUEUE STARTING AT a )
3     240 SET
4     DO DUP I ! PAUSE 2
5     +LOOP DROP ;
6
7 : INIT.QUES ( n -- INITIALIZE SPEED & DIR. QUES WITH 1ST VALUE)
8     >R NEWSPEED @ I SPEEDQ INTQ ( INIT SPEED QUEUE )
9     NEWDIR @ I ANGLEQ INTQ ( INIT DIRECTION QUEUE )
10    NEWSPEED @ 120 M* ( INIT CHECK SUMS )
11    I SPEEDCK 2!
12    NEWDIR @ 120 M* I ANGLECK 2!
13    120 R) SAMPCNT + C! ; ( ENABLE 60 MIN PEAK )
14 .S
15

```

## 507 LIST

```

0 ( DATA COUNTER ADDRESS SETUP ROUTINES 1/14/87)
1
2 : PASS.CNT ( n -- a RETURN ADDRESS OF PASSCNT FOR SENSOR n)
3   PASSCNT + ;
4
5 : CLEAR.PASSCNT ( n -- CLEAR PASS COUNT OF SENSOR n )
6   PASS.CNT 0 SWAP C! ;
7
8 : PEAK.CNT ( n -- a RETURN ADDRESS OF PEAK CNT FOR SENSOR n)
9   2* PEAKCNT + ;
10
11 : CLEAR.PEAKCNT ( n -- CLEAR PEAK COUNT OF SENSOR n )
12   PEAK.CNT 0 SWAP ! ;
13
14 : CLEAR.FAILCNT ( n -- CLEAR FAIL COUNT OF SENSOR n)
15   FAILCNT + 0 SWAP C! ; .S

```

## 508 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2 HEX
3 CREATE FLAG.TAB ( MASK TABLE USED SENSOR.STATUS)
4   1 , 2 , 4 , 10 , 20 , 40 , 100 , 200 , 400 ,
5   1000 , 2000 , 4000 ,
6
7
8
9 .S
10
11
12
13
14
15

```

## 509 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2 HEX
3 CODE sensor.status ( a,tm -- USE MASK tm TO TEST FOR ERROR COND)
4   1 POP W POP
5   SENSORSTATUS 1 TEST 0= ( match ?)
6   IF A W ) 1 TEST 0= ( no match, ack clear ?)
7   IF FFFF # 1 XOR ( yes, form clearing mask)
8   1 W ) AND ( clear bit in status)
9   THEN
10  ELSE W ) 1 TEST 0= ( matched, status clear ?)
11  IF 1 W ) OR ( set status bit)
12  1 A W ) OR ( set ack bit)
13  THEN
14  THEN NEXT
15 .S

```



## 510 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2 : SENSOR.STATUS ( n -- SETUP SENSOR STATUS WORD FORM NEW INPUT)
3     2* SENSTAT +      ( GET SENSOR n STATUS ADDR. )
4     FLAG.TAB 24 SET   ( SETUP FOR LOOP )
5     DO DUP I @        ( FETCH TEST MASK )
6     sensor.status    ( CK FOR ERROR OR ACK )
7     PAUSE 2
8     +LOOP DROP ;
9
10
11
12
13
14 .S
15

```

## 511 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2 : CLEAR.STATUS ( n -- CLEARS THE STATUS BITS SET BY BAD.DATA)
3     2* SENSTAT +      ( GET SENSOR STATUS ADDR. )
4     8192 OVER clear.status ( CLEAR OUT OF RANGE)
5     128 OVER clear.status ( CLEAR CRC FLAG)
6     8 SWAP clear.status ; ( CLEAR NO RESP. FLAG)
7
8
9
10 .S
11
12
13
14
15

```

## 512 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2 HEX
3 : CLEAR.RECOVERY ( n f -- CLR SENSOR n LONG OR SHORT RECOVERY )
4     IF 8000          ( LONG RECOVERY MASK )
5     ELSE 800         ( SHORT RECOVERY MASK )
6     THEN SWAP 2* SENSTAT + ( GET SENSOR STATUS ADDR)
7     clear.status ;    ( CLEAR RECOVERY )
8
9 : SET.RECOVERY ( n f -- SET SENSOR n LONG OR SHORT RECOVERY )
10    IF 8000          ( LONG RECOVERY MASK )
11    ELSE 800         ( SHORT RECOVERY MASK )
12    THEN SWAP 2* SENSTAT + ( GET SENSOR STATUS ADDR )
13    set.status ;    ( FLAG RECOVERY )
14
15 .S

```

## 513 LIST

```

0 ( McCLELLAN BUILT IN TEST STATUS ROUTINES 1/15/87)
1
2
3
4
5
6
7
8
9
10
11
12
13
14 .S
15

```

## 514 LIST

```

0 ( McCLELLAN VALID.PEAK? ROUTINE 1/14/87)
1
2 : VALID.PEAK? ( n -- TEST SENSOR n PEAK COUNT FOR VALID DATA)
3 DUP >R PEAK.CNT DUP @ 1+
4 24HRLIMIT MIN DUP ROT ! ( GET COUNT & SAVE)
5 DUP 24HRLIMIT ( ( COUNT ( 24 HOURS ?)
6 I 24VALID + C@ OR ( OR 24VALID SET ?)
7 IF SET.24PEAK ( 24 HR DATA INVALID)
8 60MINLIMIT ( ( COUNT ( 60 MINUTES ?)
9 I 60VALID + C@ OR ( OR 60VALID SET ?)
10 IF SET.60PEAK ( SET HRLY DATA INVALID)
11 THEN R) 1 SET.RECOVERY ( SET LONG RECOVERY )
12 ELSE DROP
13 R) 1 CLEAR.RECOVERY ( CLEAR LONG RECOVERY)
14 THEN ;
15 .S

```

## 515 LIST

```

0 ( McCLELLAN VALID.DATABASE? ROUTINE 1/14/87)
1
2 : VALID.DATABASE? ( n -- TEST VALIDITY OF SENSOR n DATABASE )
3 DUP CLEAR.FAILCNT ( CLEAR FAIL COUNTER)
4 DUP VALID.PEAK? ( CK HRLY & 24 HR PEAK DATA)
5 DUP PASS.CNT DUP C@ 1+ 10MINLIMIT MIN DUP ( UPDATE CNT)
6 ROT C! DUP 10MINLIMIT ( ( COUNT ( 10 MINUTES ?)
7 IF SET.10GROUP ( SET 10 MIN DATA INVALID)
8 DUP 2.OR.10 ( SET 2 MIN DATA ON 2 OR 10 MIN)
9 IF 2MINLIMIT ELSE 10MINLIMIT THEN (
10 IF SET.2GROUP ( SET 2 MINUTE DATA INVALID)
11 THEN 1MINLIMIT ( ( COUNT ( 1 MINUTE ?)
12 IF SET.GS ( SET GUST SPREAD INVALID)
13 THEN 0 SET.RECOVERY ( FLAG SHORT RECOVERY )
14 ELSE DROP 0 CLEAR.RECOVERY ( CLR SHORT RECOVERY )
15 THEN ; .S

```

## 516 LIST

```

0 ( PROCESS BAD DATA HANDLERS                                5/15/86)
1
2 : INC.FAILCNT ( n -- INCREMANT FAIL COUNT OF SENSOR n TO LIMIT)
3     FAILCNT + DUP C@ 1+      ( FETCH COUNT & INCREMENT)
4     10MINLIMIT MIN          ( LIMIT COUNT TO 120 )
5     SWAP C! ;               ( STORE COUNT)
6
7 : 55.STDCLEAR ( n -- CLEAR FLAGS IF SET)
8     55FLAG C@              ( CHECK 55FLAG)
9     IF DUP 55FLAG SWAP flag.settest DROP
10    THEN STDFLAG C@        ( STDFLAG )
11    IF STDFLAG SWAP flag.settest DROP
12    ELSE DROP              ( DROP n)
13    THEN ;
14 .S
15

```

## 517 LIST

```

0 ( PROCESS BAD DATA HANDLERS                                5/15/86)
1
2 : FLAG.DATA ( n -- FLAGS DATA FOR SENSOR n AS INVALID DATA )
3     DUP )R INC.FAILCNT      ( INC. INVALID DATA CTR)
4     I RESULTBUF             ( GET SENSOR DATA BUFFER ADD.)
5     TEMPBUF 32 CMOVE PAUSE  ( COPY RESULTS TO TEMP. BUFF)
6     FAILCNT I + C@ 12 )     ( LONG OR SHORT RECOVERY ? )
7     I OVER SET.INVALID     ( SET MSB OF DATA TO FLAG INVLD)
8     I OVER SET.RECOVERY    ( FLAG SHORT OR LONG RECOVERY )
9     IF I CLEAR.PEAKCNT     ( DATABASE LONG RECOVERY )
10    THEN I 55.STDCLEAR     ( CLEAR 55FLAG & STDFLAG IF SET)
11    R) CLEAR.PASSCNT ;     ( CLEAR DATA COUNTER )
12
13
14 .S
15

```

## 518 LIST

```

0 ( PROCESS BAD.DATA ROUTINE                                  5/15/86)
1
2 : BAD.DATA ( n -- FLAGS DATA FOR SENSOR n AS INVALID DATA )
3     DUP INBUFFER DUP C@    ( RETRIEVE FLAG & )
4     0 ROT C! ?DUP          ( CLEAR FLAG )
5     IF 2 =                  ( NO RESP. ERROR ?)
6     IF 8                    ( FLAG NO RESP. ERROR)
7     ELSE 128                ( FLAG CRC ERROR)
8     THEN
9     ELSE 4096                ( FLAG OVERRANGE ERROR )
10    THEN OVER 2* SENSTAT +  ( GET SENSOR n STATUS ADD)
11    set.status              ( FLAG SENSOR STATUS )
12    FLAG.DATA ;            ( SET SENSOR DATA INVALID)
13
14 .S
15

```

## 519 LIST

```

0 ( COMPUTE NEW X & Y VALUES AND SUMS                                3/12/86)
1
2 : RANGE.CHECK ( a -- x, y, f TEST INPUT FOR INVALID RANGE)
3     DUP 1+ @ ( GET X VALUE )
4     SWAP 3 + @ ( GET Y VALUE )
5     2DUP ABS 32000 )
6     SWAP ABS 32000 ) OR ; ( EITHER OUT OF RANGE )
7
8
9
10
11
12 .S
13
14
15

```

## 520 LIST

```

0 ( 1st.PROCESS INITIALIZE DATABASE ON 1ST POWER-UP 2/11/87)
1
2 : 1st.PROCESS ( n -- INITIALIZE SENSOR n DATABASE )
3     )R TEMPBUF 32 ERASE ( CLEAR TEMPORARY BUFFER)
4     PAUSE NEWSPEED @ ( GET SPEED & SCALE DOWN )
5     SCALE.DOWN SPEED ! ( INITIALIZE SPEED)
6     NEWDIR @ DUP ( GET DIRECTION )
7     DIRECTION ! ( INITIALIZE DIRECTION)
8     DIRVAR 2DUP ! 2+ ! ( INITIALIZE DIR. VARIAB.)
9     I INIT.QUES ( INITIALIZE DATA QUEUES )
10    I INIT.SUMS ( INITIALIZE RUNNING SUMS )
11    I 1 SET.INVALID ( SET DATABASE AS INVALID)
12    I 1 SET.RECOVERY ( FLAG LONG RECOVERY )
13    12345 R) 2* 1stSTART + ! ; ( SET HAVE DATA FLAG )
14 .S
15

```

## 521 LIST

```

0 ( PROCESSING TASK PROC.TIME ROUTINE                                4/5/86)
1
2
3
4
5
6
7 .S
8
9
10
11
12
13
14
15

```

## 522 LIST

```

0 ( PROCESS WIND.DATA ROUTINE      4/5/86)
1
2 : WIND.DATA ( n -- CONT. PROCESS OF WIND DATA FOR SENSOR n )
3     DUP )R GUST.SPREAD           ( GUST SPREAD )
4     I PEAK.WIND                  ( 10 MIN PEAK WIND )
5     I GUST.GUSTY PAUSE           ( GUSTS )
6     I HOUR.55                    ( HOURLY PEAK WIND )
7     I DIR.VAR                    ( DIRECTION VARIABILITY )
8     I STD.DEV                    ( DIR. STAND. DEVIATION)
9     I PEAKRESULTS
10    TEMPBUF 18 + 14 CMOVE         ( MOVE PEAK RESULTS)
11    PAUSE I VALID.DATABASE?
12    TEMPBUF I RESULTBUF 32 CMOVE PAUSE
13    I CLEAR.STATUS               ( CLR ERROR BITS )
14    O R) VALID + C! ;           ( CLR FLAG FOR PWR RESET )
15 .S

```

## 523 LIST

```

0 ( PROCESS PROC.DATA ROUTINE      4/5/86)
1
2 : PROC.DATA ( n -- PROCESS WIND DATA FOR SENSOR n )
3     )R 128 VALID I + C!         ( SET FOR PWR RESET )
4     I AVE.SPEED PAUSE           ( AVERAGE SPEED)
5     I THETA.BAR PAUSE           ( AVERAGE DIRECTION)
6     16 VALID I + C!            ( SET FOR PWR RESET )
7     R) WIND.DATA ;             ( PROCESS MORE WIND DATA )
8
9
10
11 .S
12
13
14
15

```

## 524 LIST

```

0 ( PROCESS STORE.INPUT ROUTINE    4/5/86)
1
2 : STORE.INPUT ( x y n -- CONVERTS x & y TO SPEED & DIRECTION )
3     GET.HR/MIN                  ( GET TIME OF PROCESS )
4     OVER TIMEQ ROT TAIL + !     ( UPDATE TIME QUEUE )
5     REC.POLAR                   ( CONVERT TO SPEED & DIR)
6     OVER 64 (                   ( SPEED ( 1/2 KNOT ? )
7     IF DROP 0                   ( DIRECTION = 0 )
8     ELSE ABS REV 50 +
9         100 U/ DUP NOT          ( DIR ANGLE = 0 ? )
10    IF DROP 360                 ( DIRECTION = 360 )
11    THEN
12    THEN NEWDIR ! NEWSPEED ! ; ( SAVE NEW INPUTS)
13
14
15 .S

```

## 525 LIST

```

0 ( PROCESSING TASK GET.INPUT ROUTINE          4/5/86)
1
2 HEX
3 : GET.INPUT ( n -- a f, GETS SENSOR STATUS & CKS FOR ERRORS )
4     DUP >R INBUFFER 0 OVER C!    ( CLR DATA I/P BUFF FLAG )
5     DUP 5 + @ SENSORSTATUS !    ( SAVE STAT)
6     I SENSOR.STATUS             ( UPDATE SENSOR STATUS )
7     SENSTAT R) 2* + @           ( GET SENSOR STATUS )
8     6767 AND ;                 ( ANY SENSOR PROBLEMS ? )
9
10
11
12 .S
13
14
15

```

## 526 LIST

```

0 ( PROCESS - THE MAIN WIND DATA PROCESSING WORD  4/5/86)
1
2 : PROCESS ( n -- PROCESS DATA FOR SENSOR n )
3     >R I GET.INPUT
4     IF DROP R) FLAG.DATA        ( ANY SENSOR ERRS ?)
5     ELSE RANGE.CHECK           ( FLAG BAD DATABASE )
6     IF 2DROP R) BAD.DATA       ( X,Y OUT OF RANGE ?)
7     ELSE I STORE.INPUT         ( FLAG OUT OF RANGE )
8     I 2* 1stSTART +           ( CONV & SAVE X,Y VALS)
9     @ 12345 =                 ( GET PROC. DATA FLAG )
10    IF R) PROC.DATA            ( ALREADY HAVE DATA ?)
11    ELSE I 1st.PROCESS         ( PROC FOR WIND DATA )
12    R) 1 SET.RECOVERY         ( DATA = 1ST VALUE )
13    THEN
14    THEN
15    THEN ;                      ( FLAG LONG RECOVERY)

```

## 527 LIST

```

0 ( SENSOR DATA PROCESSING TASK          4/5/86)
1
2 : PROCESS.DATA ( CKS IF ANY NEW DATA HAS ARRIVED AND PROCESSES
3     IF ITS VALID OR HANDLES IT AS INVALID DATA)
4     BEGIN ?.#SENSORS 1+ 0 ( CHECK ONLY THE CONFIRD SNSR)
5     DO I INBUFFER C@ ?DUP ( GET FLAG AT START OF BUFFER)
6     IF 1 =
7     IF I PROCESS              ( SET SENSOR # AND PROCESS)
8     ELSE I BAD.DATA          ( FLAG BAD DATA )
9     THEN
10    THEN PAUSE
11    LOOP
12    AGAIN ;
13 .S
14
15

```

## 528 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 529 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 530 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 HEX
3 : (TYPE) ( INTER-ASSY VECTOR TYPE ROUTINE )
4     PTR @ C@ 1 PTR +!           ( GET FIRST CHARACTER )
5     CTR @ 1- CTR !             ( DECREMENT CTR )
6     comm/data OUTPUT STOP ;    ( O/P CHAR )
7
8 : (EXPECT) ( INTER-ASSY VECTOR EXPECT ROUTINE )
9     STOP ;
10
11 CODE act.carr ( n -- ACTIVATES/DEACTIVATES CARRIER BY RTS O/P )
12     CLI 5 #B 0 MOV comm/csr OUT ( POINT TO REG 5 )
13     0 POP comm/csr OUT STI NEXT
14
15 .S

```



## 534 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : TEST.UARTA ( TESTS INTER-ASSY TRANSMITTER LOOPBACK )
3     16 I/RSTATUS          ( SETUP STATUS )
4     CHARBUFFA C@         ( GET 1ST CHAR LOOPBACKED)
5     16 =                  ( IS IT 'DLE' CHAR ? )
6     IF clear.status      ( FLAG NO XMIT ERR )
7         0 UARTAERR C!    ( CLEAR ERROR CTR )
8     ELSE UARTAERR C@ 1+  ( INC. ERROR CTR )
9         3 MIN DUP        ( NO MORE THAN 3 )
10        UARTAERR C! 3 =  ( ERRORS = 3 ? )
11        IF set.status    ( FLAG XMIT ERROR )
12        ELSE clear.status ( FLAG NO XMIT ERR )
13        THEN
14        THEN 0 CHARBUFFA C! ;
15 .S

```

## 535 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 HEX
3 CODE crc-16 ( n a -- bcc, RETURNS CRC-16 BCC FOR n CHARS AT a )
4     W POP 0 # 0 MOV 0 # 2 MOV
5     BEGIN W ) 2 MOV B 2 PUSH 8 # 1 MOV
6     BEGIN 2 POP 2 PUSH 0 2 XOR 1 # 2 AND 0=
7     IF 0 SHR
8     ELSE 2 SHR 0 RCR 2001 # 0 XOR
9     THEN 2 POP 2 SHR B 2 PUSH
10    LOOP 2 POP W INC 2 POP 2 DEC 2 PUSH 0=
11    UNTIL 2 POP 0 PUSH NEXT      ( RETURN CRC-16 BCC )
12
13 .S
14
15

```

## 536 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : CRC-16? ( -- f, DETER. IF CRC-16 BCC SENT IS VALID )
3     ETXPTR @ STXPTR @ 2DUP -      ( SETUP COUNT & PTR )
4     SWAP 1+ crc-16 PAUSE          ( CALCULATE CRC-16 )
5     SWAP 1+ @                      ( GET CRC-16 IN BUFFER )
6     = ;                             ( RETURNS TRUE IF = )
7
8 : ?BUFF.END ( -- a f, DETERMINES IF AT END OF COMMBUFF )
9     ETXPTR @ DUP 2+ SWAP COMMBUFF 68 + U< NOT ;
10
11 : !CHAR ( a c -- a+1, STORES CHAR c AT a )
12     OVER C! 1+ ;
13
14 : FLG.BAD.DAT ( f n -- FLAGS REC. BAD OR NO RESP FOR SENSOR n )
15     INBUFFER C! ;                .S

```

## 537 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : COMM.XMIT ( n -- TRANSMIT COMMAND WITH OPCODE n REQUEST )
3     COMMBUFF 1+ 16 !CHAR      ( STORE DLE CHAR )
4     2 !CHAR                   ( STORE STX CHAR )
5     SWAP !CHAR                ( STORE OPCODE )
6     POLLCNT C@ !CHAR          ( STORE POLL COUNT )
7     16 !CHAR 3 !CHAR          ( STORE DLE & ETX CHARS )
8     4 COMMBUFF 3 + crc-16     ( CALCULATE CRC-16 )
9     SWAP !                     ( STORE CRC-16 CKSUM )
10    CARR.ON                    ( TURN CARRIER ON )
11    COMMBUFF 1+ 8 TYPE        ( SEND POLL REQUEST )
12    TEST.UARTA                ( I/A XMITTER TEST )
13    CARR.OFF ;                ( TURN CARRIER OFF )
14
15 .S

```

## 538 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : FIND.ETX ( a -- f, FINDS POSITION OF "ETX" CHAR IN COMMBUFF )
3     PAUSE 0 ETXPTR !
4     COMMBUFF 68 + SWAP        ( START AT a )
5     DO I C@ 16 =              ( DLE CHAR ? )
6         IF I 1+ C@ 3 =        ( ETX CHAR ? )
7             IF I 1+ ETXPTR !   ( POINT TO ETX CHAR )
8                 1 LEAVE        ( FOUND ETX CHAR )
9             ELSE 2              ( SKIP NEXT CHAR )
10            THEN
11            ELSE 1              ( TRY NEXT CHAR )
12            THEN
13            +LOOP ETXPTR @ PAUSE ; ( RETURN ETX POSITION )
14
15 .S

```

## 539 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : FIND.STX ( a -- f, FINDS POSITION OF "STX" CHAR IN COMMBUFF )
3     PAUSE 0 STXPTR !
4     COMMBUFF 68 + SWAP        ( START AT a )
5     DO I C@ 16 =              ( DLE CHAR ? )
6         IF I 1+ C@ 2 =        ( STX CHAR ? )
7             IF I 1+ STXPTR !   ( POINT TO STX CHAR )
8                 1 LEAVE        ( FOUND STX CHAR )
9             ELSE 2              ( SKIP NEXT CHAR )
10            THEN
11            ELSE 1              ( TRY NEXT CHAR )
12            THEN
13            +LOOP STXPTR @ PAUSE ; ( RETURN STX POSITION )
14
15 .S

```

## 540 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : FIND.MESS ( -- f, LOCATES MESSAGE & CHECKS CRC-16 CKSUM )
3     COMMBUFF 1+ 3 0
4     DO FIND.STX ?DUP           ( FIND DLE/STX CHARS ? )
5     IF 1+ FIND.ETX           ( FIND DLE/ETX CHARS ? )
6     IF CRC-16?               ( CRC-16 VALID ? )
7     IF LEAVE 1                ( FLAG VALID CRC-16 )
8     ELSE ?BUFF.END           ( END OF BUFFER ? )
9     IF LEAVE NOT THEN        ( FLAG INVALID CRC-16 )
10    THEN
11    ELSE LEAVE 0              ( FLAG NO DLE/ETX CHARS )
12    THEN
13    ELSE LEAVE 0              ( FLAG NO DLE/STX CHARS )
14    THEN PAUSE
15    LOOP ;                    .S

```

## 541 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : GOOD.DATA ( n -- GETS SENSOR n RESPONSE DATA FROM COMMBUFF )
3     INBUFFER 1+                ( POINT TO SENSOR PROC BUFF )
4     STXPTR @ 3 +                ( POINT TO DATA IN MESSAGE )
5     OVER 6 SET                  ( SET DO-LOOP )
6     DO 2 0                      ( GET DATA LSB & MSB )
7     DO DUP C@ 16 =              ( DATA CHAR = DLE ? )
8     IF 1+                       ( SKIP DLE CHAR )
9     THEN DUP C@ SWAP 1+         ( GET DATA FROM COMMBUFF )
10    LOOP SWAP 256 * ROT +        ( COMBINE DATA BYTES )
11    I ! PAUSE 2                  ( STORE IN SENSOR PROC BUFF )
12    +LOOP DROP 1-
13    1 SWAP C! ;                  ( FLAG PROCESS DATA IN BUFF )
14
15 .S

```

## 542 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : INC.POLLCNT ( INCREMENTS SENSORS POLL COUNT # )
3     POLLCNT C@ 1+ 16
4     MOD POLLCNT C! ;           ( INC. POLL CTR )
5
6 : NEXT.POLL#? ( -- f, DETER IF POLL # IS NEXT IN SEQUENCE )
7     STXPTR @ 2+ C@             ( GET POLL # FROM COMMBUFF )
8     POLLCNT C@ 1+              ( GET EXISTING POLL COUNT )
9     16 MOD = ;                 ( LEAVES TRUE IF NEXT COUNT )
10
11 : SAME.POLL#? ( -- f, DETER. IF SAME POLL # AS EXISTING )
12    STXPTR @ 2+ C@             ( GET POLL # FROM COMMBUFF )
13    POLLCNT C@ = ;             ( LEAVES TRUE IF SAME COUNT )
14
15 .S

```

## 543 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : CLR.CRC16 ( CLEARS CRC-16 ERROR COUNTER )
3     0 CRC16CNT C! ;
4
5 : CRC.ERR? ( CHKS & FLAGS IF COMMUNICATION HAS CRC-16 ERRORS )
6     CRC16CNT C@ 1+ DUP           ( INC CRC-16 ERROR CTR )
7     CRC16CNT C! 6 =             ( CRC-16 ERR CTR = 6 ? )
8     IF ?.#SENSORS 1+ 0         ( GET # OF SENSORS )
9     DO 3 I FLG.BAD.DAT PAUSE   ( FLAG CRC-16 ERROR )
10    LOOP CLR.CRC16             ( CLR CRC-16 ERROR CTR )
11    THEN ;
12
13 .S
14
15

```

## 544 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : DETER.POLL# ( DETER. ACTUAL POLL CNT & FLAGS SENSOR BAD DATA )
3     STXPTR @ 2+ C@             ( GET SENSOR POLL # )
4     POLLCNT C@ 1+ OVER 2DUP ) ( POLL CNT ) POLL # ? )
5     IF 16 +
6     THEN SWAP -                ( DETER DIFF IN POLLS )
7         ?.#SENSORS 1+         ( GET # OF SENSORS )
8         DUP >R MIN            ( DIFF (<= # OF SENS. )
9         STXPTR @ 1+ C@ 48 AND ( GET SENSOR # & )
10        16 / DUP ROT -        ( DETER BAD SENSORS )
11        DO 2 I DUP 0(
12        IF J +
13        THEN FLG.BAD.DAT PAUSE 1 ( FLAG NO RESP )
14        +LOOP
15        R) DROP 1- POLLCNT C! ; ( POLLCNT = POLL # - 1 )

```

## 545 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : XMIT.TAKE ( TRANSMITS TAKE OVER COMMAND )
3     ACT.SENSOR? 64 *           ( ACT. SENSOR # AND )
4     15 OR COMM.XMIT ;        ( TAKE OVER OPCODE )
5
6 : ?MAST.CONTRL ( SENDS TAKE OVER COMMAND IF MASTER ASSY )
7     CONFIG C@ 8 AND           ( MASTER ASSY ? )
8     IF STXPTR @ 1+ C@        ( GET SEN. RESP # )
9         48 AND 16 /          ( GET # OF SENSORS )
10        ?.#SENSORS =         ( SAME SENSOR # ? )
11        IF INC.POLLCNT XMIT.TAKE
12        ELSE 0 POLLFLAG C!   ( DON'T SEND POLL REQ. )
13        THEN
14        THEN ;
15 .S

```

## 546 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : SEN.RES ( ROUTINE TO HANDLE SENSOR POLL RESPONSES )
3     NEXT.POLL#? NOT          ( NOT IN NEXT POLL # SEQ ? )
4     IF DETER.POLL#          ( FLAG MISSING SENSOR DATA )
5     THEN STXPTR @ 1+ C@ 48  ( GET SENSOR # IN COMMBUFF )
6         AND 16 / GOOD.DATA  ( STORE DATA IN PROC BUFF )
7         INC.POLLCNT
8         ?MAST.CONTRL ;      ( CK TO SEND TAKEOVER COM. )
9
10 : SEN.RES2 ( ROUTINE TO HANDLE SENSOR REPOLL RESPONSES )
11     SAME.POLL#? NOT        ( NOT SAME POLL # AS EXISTING ? )
12     IF SEN.RES             ( GET DATA FORM COMMBUFF )
13     THEN ;
14
15 .S

```

## 547 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : SEN.POLL ( ROUTINE TO HANDLE SENSOR POLL REQUESTS )
3     NEXT.POLL#? NOT        ( NOT NEXT POLL # SEQUENCE ? )
4     IF DETER.POLL#        ( FLAG MISSING SENSOR DATA )
5     THEN CONFIG C@ 12 AND 0= ( BACKUP ASSY ? )
6         IF 0 CONTRLFLAG C!  ( STOP SENSOR POLLING )
7         0 FAILCTR C!        ( CLR TIMEOUT FAIL CTR )
8     THEN ;
9
10 : SEN.REPOLL ( ROUTINE TO HANDLE SENSOR REPOLL REQUESTS )
11     SAME.POLL#? NOT        ( NOT SAME POLL # AS EXISTING ? )
12     IF SEN.POLL           ( CK IF POLL # NEXT IN SEQ )
13     THEN ;
14
15 .S

```

## 548 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : POLL/RESPONSE ( n -- DETER. IF RECP. SENSOR POLL OR RESPONSE )
3     CASE
4     1 OF SEN.POLL END OF ( SENSOR POLL REQUEST )
5     2 OF SEN.REPOLL END OF ( SENSOR REPOLL REQUEST )
6     3 OF SEN.RES END OF ( SENSOR POLL RESPONSE )
7     4 OF SEN.RES2 END OF ( SENSOR REPOLL RESPONSE )
8     ENDCASE ;
9
10 : MASTER.REC ( PUTS BACKUP INTO REGULAR MODE )
11     0 CONTRLFLAG C!      ( STOP SENSOR POLLING )
12     0 FAILCTR C!        ( CLR TIMEOUT FAIL CTR )
13     POLLCNT C@ 1- DUP 0 ( DEC. POLLCNT ( 0 ? ) )
14     IF 16 +             ( 16 MOD )
15     THEN POLLCNT C! ;    ( POLLCNT = CNT -1 ) .S

```

## 549 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 HEX
3 : BACKUP.CONT? ( n -- n, CKS IF BACKUP ASSY HAS COMM. CONTROL )
4         DUP 8 AND ( BACKUP OPCODE ? )
5         400 I/RSTATUS ROT ( SETUP STATUS )
6         IF set.status ( FLAG NO MASTER )
7         ELSE clear.status ( FLAG MASTER OKAY )
8         THEN ;
9
10 : TAKE.OVER ( n -- PUTS BACKUP ASSY INTO NORMAL MODE )
11         DROP CONFIG C@ 0C AND 0= ( BACKUP ASSY ? )
12         IF MASTER.REC ( CLR ERROR CTRS )
13         400 I/RSTATUS clear.status ( CLR LOSS OF MASTER )
14         40 I/RSTATUS clear.status ( CLR CARR. TIMEOUT )
15         THEN ; .S

```

## 550 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 HEX
3 : NEW.ACTSEN ( n -- DETER. IF NEW ACTIVE SENSOR FROM OPCODE n )
4         40 / DUP ( OPCODE ACT. SENSOR # )
5         ACT.SENSOR? = ( SAME AS EXISTING SEN # )
6         IF DROP
7         ELSE SENSOR# C@ FC AND
8         OR SENSOR# C! ( SAVE NEW ACTIVE SEN # )
9         CONFIG C@ 10 AND 0= ( RECORDER ASSY ? )
10        IF ALARM.ON ( SOUND RECORDER ALARM )
11        THEN 1 DISPFLAG C! ( FLAG NEW DISPLAY )
12        THEN ;
13
14 .S
15

```

## 551 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : CASE.RECP ( DETERMINES COMM. RECEPTION REQUEST VIA OPCODE )
3         CLR.CRC16 ( CLR CRC-16 ERROR CTR )
4         STXPTR @ 1+ C@ ( GET OPCODE FROM COMMBUFF )
5         DUP NEW.ACTSEN ( CK FOR NEW ACTIVE SEN. # )
6         15 AND DUP 15 = ( MASTER TAKEOVER COMMAND ? )
7         IF TAKE.OVER
8         ELSE BACKUP.CONT? ( CK IF BACKUP HAS CONTROL )
9         7 AND ( GET COMMAND REQUEST )
10        POLL/RESPONSE
11        THEN ;
12
13
14 .S
15

```

## 552 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : ?BACK.UP ( CKS IF BACKUP ASSY & IF TO START XMITING POLL REQ )
3     CONFIG C@ 12 AND 0=           ( BACKUP ASSY ? )
4     IF FAILCTR C@ 1+ DUP         ( INC. TIMEOUT FAIL CTR )
5     FAILCTR C! 2 =              ( FAILED TWICE AND )
6     IF INC.POLLCNT              ( POLL CNT = CNT +1 )
7     1 CONTRLFLAG C!            ( FLAG BACKUP TO POLL )
8     55 TIMEOUT C!              ( SET FOR 5.5 SEC TIMEOUT)
9     THEN
10    THEN ;
11
12 .S
13
14
15

```

## 553 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : BAD.DATA ( FLAGS BAD DATA FOR ALL SYSTEM SENSORS )
3     32 I/RSTATUS set.status      ( FLAG 5 SEC TIMEOUT ERR)
4     ?.#SENSORS 1+ DUP 0         ( DETER # OF SENSORS )
5     DO 2 I FLG.BAD.DAT PAUSE    ( FLAG NO RESP. ERROR )
6     LOOP POLLCNT C@ +          ( POLL CNT = POLL CNT + )
7     16 MOD POLLCNT C!          ( # OF SENSORS )
8     CLR.CRC16                  ( CLR CRC-16 ERROR CTR )
9     ?BACK.UP                   ( CK IF BACKUP ASSY )
10    ACTCTR C@ 3 MIN             ( GET CARR. DETECT CTR )
11    DUP 3 = 64 I/RSTATUS ROT    ( 3 IN A ROW ? )
12    IF set.status               ( FLAG CARR. TIMEOUT ERR)
13    ELSE clear.status           ( CLR CARR. TIMEOUT ERR )
14    THEN ACTCTR C! ;
15 .S

```

## 554 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : PROC.RECP ( PROCESS RECEPTION IN COMMBUFF )
3     COMMBUFF C@ 2 =            ( COMM. TIMEOUT ? )
4     IF BAD.DATA                ( FLAG SENSORS BAD DATA)
5     ELSE 32 I/RSTATUS
6     clear.status                ( CLR 5 SEC TIMEOUT ERR )
7     CNT @ 7 )                  ( HAVE POSSIBLE MESSAGE?)
8     IF FIND.MESS                ( VALID RECEPTION ? )
9     IF CASE.RECP                ( DETER. RECP. REQUEST )
10    ELSE CRC.ERR?              ( DETER. IF CRC-16 ERR.)
11    THEN
12    THEN
13    THEN ;
14 .S
15

```

## 555 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : GET.DATA ( GETS SENSOR RESPONSE DATA FROM COMMBUFF )
3     32 I/RSTATUS clear.status ( CLR 5 SEC TIMEOUT ERR )
4     STXPTR @ 1+ C@ 48 AND      ( GET SENSOR # FROM OPCODE )
5     16 / POLLSEN# C@ =        ( SAME SENSOR POLLED ? )
6     IF POLLSEN# C@ GOOD.DATA ( GET & STORE SENSOR DATA )
7         0 POLLSEN# C@
8         BADPOLLS + C!         ( CLEAR FAIL POLL CTR )
9         0                     ( FLAG RECEPTION PROCESSED )
10        ELSE 2                ( FLAG NO RESPONSE ERROR )
11        THEN ;
12
13
14 .S
15

```

## 556 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : PROC.RES ( -- f, PROCESSES SENSOR RESPONSE FOR VALID DATA )
3     COMMBUFF C@ 1 =          ( REC. A RESPONSE ? )
4     IF FIND.MESS            ( LOCATE RESP IN COMMBUFF )
5     IF GET.DATA             ( STORE SENSOR RESP. DATA )
6     ELSE 3                  ( FLAG CRC-16 ERROR )
7     THEN
8     ELSE 2                  ( FLAG NO RESP. ERROR )
9     THEN ;
10
11
12
13 .S
14
15

```

## 557 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : GET.RESP ( -- f, AWAITS FOR SENSOR POLL RESP )
3     COMMBUFF 70 ERASE       ( CLEAR COMM. I/P BUFFER )
4     1 RESPFLAG C!          ( FLAG EXPECTING SENSOR RESP )
5     3 TIMEOUT C!           ( SETUP TIMEOUT FOR 300 mS )
6     COMMBUFF 1+ 69 EXPECT  ( SETUP FOR RESPONSE )
7     0 RESPFLAG C!         ( CLEAR EXPECTING RESP FLAG )
8     55 TIMEOUT C!         ( SET TIMEOUT TO 5.5 SEC )
9     PROC.RES ;             ( PROCESS SENSOR RESPONSE )
10
11
12
13 .S
14
15

```

## 558 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : FORMAT.CODE ( n -- n, FORMATS SENSOR POLL OR REPOLL REQ CODE )
3     IF 2 ( REPOLL )
4     ELSE 1 ( POLL )
5     THEN POLLSSEN# C@ 16 * OR ( SENSOR ID # )
6         ACT.SENSOR? 64 * OR ( ACT. SENSOR # )
7         CONFIG C@ 8 AND DUP ( GET CONF. MODE )
8         1024 I/RSTATUS ROT ( MASTER ASSY ? )
9         IF clear.status ( CLR LOSS OF MASTER )
10        ELSE set.status ( FLAG LOSS OF MASTER )
11        THEN 8 XOR OR ; ( MASTER/BACKUP CODE )
12
13 .S
14
15

```

## 559 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : SEND.POLL ( n -- SENDS SENSOR POLL OR REPOLL REQUEST )
3     FORMAT.CODE ( FORMAT REQUEST OPCODE )
4     COMM.XMIT ; ( XMIT POLL OR REPOLL )
5
6 : POLL.FAIL ( INCREMENTS FAIL POLL COUNTER FOR BAD SENSOR )
7     POLLSSEN# C@ BADPOLLS + DUP ( GET SENSOR CTR )
8     C@ 1+ 2 MIN SWAP C! ; ( INC. FAIL POLL CTR )
9
10
11
12
13 .S
14
15

```

## 560 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : POLL.RTN ( n -- POLLS AND PROCESSES DATA FOR SENSOR n )
3     DUP POLLSSEN# C! ( TEMP FOR SENSOR # )
4     BADPOLLS + C@ 2 = ( FAILED TWICE BEFORE ? )
5     IF 1 ELSE 3 THEN 0 ( SETUP FOR n POLL TRIES )
6     DO I SEND.POLL ( XMIT SENSOR POLL REQ )
7     GET.RESP ?DUP ( POLL RESP GOOD ? )
8     IF I I' 1- = ( LAST TRY ? )
9     IF POLL.FAIL POLLSSEN# C@
10    FLG.BAD.DAT ( FLAG BAD RESPONSE )
11    ELSE DROP
12    THEN
13    ELSE LEAVE ( REC. RESPONSE )
14    THEN PAUSE
15    LOOP ; .S

```

## 561 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : POLL.TASK ( POLLS SENSORS FOR WIND DATA )
3     0 POLLFLAG C!           ( CLEAR POLL SENSOR FLAG )
4     ?.#SENSORS 1+ 0        ( DETER # OF SENSORS )
5     DO I POLL.RTN          ( TO POLL )
6         INC.POLLCNT PAUSE
7     LOOP 55 TIMEOUT C! ;    ( RESET FOR 5.5 SEC TIMEOUT)
8
9
10
11 .S
12
13
14
15

```

## 562 LIST

```

0 ( INTER-ASSY COMMUNICATION TASK ROUTINE )
1
2 : INTER.ASSY ( HANDLES COMMUNICATION WITH SENSOR )
3     BEGIN COMMBUFF 70 ERASE
4         COMMBUFF 1+ 69 EXPECT      ( SETUP FOR RECP. )
5         55 TIMEOUT C!           ( RESET TIME OUT TO 5.5 SEC. )
6         POLLFLAG C@           ( TIME FOR SENSOR POLLS ? )
7         IF POLL.TASK           ( START SENSOR DATA POLLING )
8         ELSE PROC.RECP         ( CHK FOR VALID RECEPTION )
9         THEN PAUSE
10        AGAIN ;
11
12 .S
13
14
15

```

## 563 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

564 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

565 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

566 LIST

- 0 ( INTENTIONALLY BLANK SCREEN )
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

## 567 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 568 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 569 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK           4/28/89 )
1
2 HEX
3 LABEL 10sdelay
4     OF #B 6SECCTR TEST 0= NOT           ( 6 SEC CTR ACTIVE ? )
5     IF 6SECCTR DEC B 0) NOT           ( DEC. CTR, = 0 ? )
6         0 #B PRINTFLAG MOV           ( HOLD NEW PRINTOUT )
7         0 #B DISPFLAG MOV           ( HOLD DISPLAY UPDATE )
8     IF 1 #B PRINTFLAG MOV           ( PRINT 55 MIN. DATA )
9         1 #B DISPFLAG MOV           ( UPDATE DISPLAY DATA )
10    THEN
11    THEN RET                           .S
12
13
14
15

```

## 570 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK           4/28/89 )
1
2 HEX
3 LABEL (RTC) ( REAL TIME CLOCK INTERRUPT HANDLER )
4     clock/intr IN                          ( GET INTER. STATUS )
5     10 #B 0 TEST 0= NOT                    ( MIN. INTER ? )
6     IF 0 1 MOV clock # 2 MOV (2) IN        ( LATCH DATA )
7         MINUTE #B 2 ADD (2) IN            ( GET MIN. CTR)
8         37 #B 0 CMP 0=                   ( MINUTE = 55 ?)
9     IF OF #B 55FLAG MOV                   ( SET 55FLAG )
10        06 #B 6SECCTR MOV                ( SET 6 SEC COUNTER )
11    THEN OF #B STDFLAG MOV               ( UPDATE STD DEV. )
12        10 #B CONFIG TEST 0=            ( RECORDER ASSY ? )
13    IF 1 #B PRINTFLAG MOV               ( PRINT DATA )
14        1 #B DISPFLAG MOV               ( UPDATE DATA DIPLAY )
15    THEN 1 0 MOV                          .S

```

## 571 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK           23/NOV/86 )
1
2 HEX
3     THEN 08 #B 0 TEST 0= NOT              ( SECOND INTERRUPT ? )
4     IF 10 #B CONFIG TEST 0= NOT         ( INDICATOR ASSY ? )
5     IF 0 #B SENSOR# TEST 0= NOT        ( MOMENTARY DISP ? )
6     IF 1 #B STATDISFLAG TEST 0=      ( NOT IN STATUS DISP ? )
7     IF 1MINCTR DEC B 0) NOT           ( 1 MINUTE TIMER ENDED ? )
8     IF OF #B SENSOR# AND              ( DISABLE MOMENTARY DISPLAY )
9     1 #B DISPFLAG MOV                 ( UPDATE DISPLAY DATA )
10    THEN
11    THEN
12    THEN
13    THEN 5SECCTR DEC B 0) NOT         ( 5 SEC. TIMER = 0 ? )
14 .S
15

```

## 572 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK           23/NOV/86 )
1
2 HEX
3     IF 5 #B 5SECCTR MOV                ( RESET 5 SEC. TIMER )
4     1 #B AWDSFLAG MOV                  ( SET O/P THRU AWDS PORT FLAG )
5     10 #B CONFIG TEST 0= NOT          ( INDICATOR ASSY ? )
6     IF 1 #B DISPFLAG MOV              ( UPDATE DISPLAY DATA )
7     THEN 1 #B CONTRLFLAG TEST 0= NOT  ( COMM CONTROL ? )
8     IF 8 #B EXTSTATAFLG TEST 0=      ( NO CARRIER ? )
9     IF 1 #B POLLFLAG MOV              ( START SENSOR POLLS )
10    WAKE # U ) MOV                    ( WAKE INTER-ASSY TASK )
11    0 #B ACTCTR MOV                   ( CLR CD COUNTER )
12
13
14 .S
15

```

## 573 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK                4/28/89 )
1
2 HEX
3           ELSE 0A #B TIMEOUT MOV                ( RESET TIME OUT CTR )
4           ACTCTR INC B                          ( INC. CD COUNTER )
5           THEN
6           THEN
7           THEN 10sdelay CALL                    ( CHECK FOR DELAY )
8
9 .S
10
11
12
13
14
15

```

## 574 LIST

```

0 ( REAL TIME CLOCK INTERRUPT TASK                23/NOV/86 )
1
2 HEX
3           THEN 04 #B 0 TEST 0= NOT              ( 100 mSEC INTERRUPT ? )
4           IF 500MSCTR DEC B 0) NOT              ( 500 mSEC TIMER FIN. ? )
5           IF 5 #B 500MSCTR MOV                  ( RESET 500 mSEC TIMER )
6           1 #B REFRESHFLAG MOV                  ( REFRESH DISPLAY )
7           THEN TIMEOUT DEC B 0) NOT             ( COMM. TIMEOUT ? )
8           IF 2 #B COMMBUFF MOV                   ( FLAG COMM. TIMEOUT )
9           WAKE # U ) MOV                         ( WAKE INTER-ASSY TASK )
10          THEN FF #B CARTIMER TEST
11          0= NOT                                  ( CARRIER TIMER ON ? )
12          IF CARTIMER DEC B                       ( DEC. TIMEOUT CTR )
13          THEN
14          THEN RET
15 .S

```

## 575 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 576 LIST

```

0 ( FRONT PANEL SWITCH SUBROUTINE HANDLER      23/NOV/86 )
1
2 HEX
3 LABEL moment.sw ( SUBRTN. TO HANDLE SENSOR/DISPLAY SW. SELECT )
4   0 INC 04 #B 1 MOV 0 SHL V ( shift to upper nibble )
5   10 #B CONFIG TEST 0= NOT ( indicator assembly ? )
6   IF 1 #B STATDISFLAG TEST 0= ( not in status mode ? )
7   IF F0 #B 1 MOV SENSOR# 1 AND B
8   0 1 CMP B 0= ( disable momentary mode ? )
9   IF 0 #B 0 MOV ( setup for active sen. disp. )
10  ELSE 3B #B 1MINCTR MOV ( start 1 min momentary ctr )
11  THEN
12  THEN
13  THEN OF #B SENSOR# AND 0 SENSOR# OR B ( new display )
14 .S
15

```

## 577 LIST

```

0 ( FRONT PANEL SWITCH SUBROUTINE HANDLER      23/NOV/86 )
1
2 HEX
3   10 #B CONFIG TEST 0= NOT ( indicator assembly ? )
4   IF 1 #B DISPFLAG MOV ( flag for new display )
5   THEN RET
6
7
8
9 .S
10
11
12
13
14
15

```

## 578 LIST

```

0 ( FRONT PANEL SWITCH SUBROUTINE HANDLER      23/NOV/86 )
1
2 HEX
3 LABEL ack.error ( SUBRTN. TO HANDLE CLR. STATUS SW. SELECTION )
4   0 # I/RACK MOV ( CLEAR IND/REC ACKNOWLEDGE )
5   0 # SENACK MOV ( CLEAR SENSOR 1 ACKNOWLEDGE )
6   0 # SENACK 2+ MOV ( CLEAR SENSOR 2 ACKNOWLEDGE )
7   0 # SENACK 4 + MOV ( CLEAR SENSOR 3 ACKNOWLEDGE )
8   0 # SENACK 6 + MOV ( CLEAR SENSOR 4 ACKNOWLEDGE )
9   RET
10
11 .S
12
13
14
15

```

## 579 LIST

```

0 ( FRONT PANEL SWITCH SUBROUTINE HANDLER      23/NOV/86 )
1
2 HEX
3 LABEL clock.switch ( SUBRTN. TO HANDLE CLOCK SW. SELCTIONS )
4     C8 #B 0 CMP 0= ( set/run switch?)
5     IF 40 #B CLOCKFLAG XOR ( toggle set/run )
6         C0 #B CLOCKFLAG AND ( clear up, down & field )
7     ELSE D0 #B 0 CMP 0= ( field select switch ?)
8         IF CLOCKFLAG INC B ( inc field mod 6 )
9             07 # 0 MOV CLOCKFLAG 0 AND B 6 # 1 MOV 1 DIV B
10            0 HI 0 MOV B F0 # 0 OR 0 CLOCKFLAG AND B
11            ELSE D8 #B 0 CMP 0= ( up switch ?)
12 .S
13
14
15

```

## 580 LIST

```

0 ( FRONT PANEL SWITCH INTERRUPT HANDLER      23/NOV/86 )
1
2 HEX
3     IF 20 #B CLOCKFLAG OR ( set up flag)
4     ELSE E0 #B 0 CMP 0= ( down flag ?)
5         IF 10 #B CLOCKFLAG OR ( set down flag)
6             THEN
7                 THEN
8                 THEN
9                 THEN RET
10
11
12
13 .S
14
15

```

## 581 LIST

```

0 ( FRONT PANEL SWITCH INTERRUPT HANDLER      23/NOV/86 )
1
2 HEX
3 LABEL (KEYBOARD) ( 8279 KEYBOARD INTERRUPT HANDLER )
4     fpi/data IN 0 1 MOV fpi/csr IN 30 #B 0 TEST 0= NOT
5     IF RET ( RETURN FOR UNDER/OVER FLOW CONDITION )
6     THEN 1 0 MOV 68 #B 0 CMP 0= ( status & lamp test ?)
7     IF OF #B SENSOR# AND 1 #B STATDISFLAG XOR ( toggle bit)
8         0= NOT ( status display mode ?)
9     IF 10 #B SENSOR# OR ( no, setup for sensor 1)
10    ELSE 10 #B CONFIG TEST 0= ( recorder assembly ?)
11    IF 10 #B SENSOR# OR ( set recorder to normal display)
12    THEN
13    THEN 1 #B DISPFLAG MOV ( set DISPLAYFLAG )
14    80 #B CLOCKFLAG AND ( CLEAR ALL CLOCK FUNCTIONS)
15 .S

```

## 582 LIST

```

0 ( FRONT PANEL SWITCH INTERRUPT HANDLER      23/NOV/86 )
1
2 HEX ( alarm reset switch selection )
3     ELSE C8 #B 0 CMP 0 ( sensor/display or alarm reset ?)
4     IF 04 #B 0 TEST 0= NOT ( alarm reset ?)
5     IF 10 #B CONFIG TEST 0= ( recorder assy ?)
6     IF 08 #B LEDSTATE OR LEDSTATE 0 MOV B
7     mcs/reg OUT ( reset recorder alarm )
8     THEN
9     ELSE 40 #B 0 TEST 0= NOT ( sensor/display select ?)
10
11
12
13
14
15

```

## 583 LIST

```

0 ( FRONT PANEL SWITCH INTERRUPT HANDLER      23/NOV/86 )
1
2 HEX ( sensor/display or active switch selection )
3     IF LABEL momentary.sel
4     moment.sw CALL
5     ELSE CONFIG 1 MOV B 18 #B 1 AND ( get system config )
6     18 #B 1 CMP 0= ( master ind. assy?)
7     IF 1 #B STATDISFLAG TEST 0= ( not in stat. mode?)
8     IF 0F #B 0 AND CONFIG 1 MOV B ( get # of sensors )
9     3 #B 1 AND 1 0 CMP 0) NOT ( legal sensor # ? )
10    IF 0 SENSOR# MOV B ( store new active sen.)
11    1 #B DISPFLAG MOV ( show new display )
12    ELSE momentary.sel JMP ( select sensor status disp)
13    THEN
14    ELSE momentary.sel JMP ( select sensor status disp)
15    THEN

```

## 584 LIST

```

0 ( FRONT PANEL SWITCH INTERRUPT HANDLER      23/NOV/86 )
1
2 HEX ( clear status )
3     ELSE momentary.sel JMP ( not master assy)
4     THEN
5     THEN
6     THEN
7     ELSE E8 #B 0 CMP 0= ( clear status switch ?)
8     IF ack.error CALL ( clear & acknowledge sys. errors )
9     ELSE clock.switch CALL ( ck if clock switch )
10    THEN
11    THEN
12    THEN RET
13
14 .S
15

```



## 588 LIST

```

0 ( AWDS INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL <INTRO> ( UART B TBRE INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH          ( SAVE REGISTER )
5     'AWDS.TASK # U MOV
6     CTR U) 0 MOV 0 0 DR 0)              ( HAVE ANY CHARS TO XMIT ? )
7     IF PTR U) I XCHG LODS B PTR U) I XCHG
8         awds/data OUT CTR U) DEC
9     ELSE WAKE # STATUS U) MOV           ( WAKE TASK ON TC )
10        28 #B 0 MOV awds/csr OUT      ( RESET ON LAST TBRE )
11    THEN ireturn JMP
12 .S
13
14
15

```

## 589 LIST

```

0 ( AWDS INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL <INTR1> ( UART B EXTERNAL STATUS INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH
5     awds/csr IN                          ( GET EXT. STATUS FROM RRO B )
6     10 #B 0 MOV awds/csr OUT            ( RESET EXT/STATUS INTERRUPT )
7     ireturn JMP
8
9
10
11 .S
12
13
14
15

```

## 590 LIST

```

0 ( AWDS INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL <INTR2> ( UART B DATA READY INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH          ( SAVE REGS )
5     'AWDS.TASK # U MOV awds/data IN      ( GET CHAR )
6     80 #B CTR U) TEST 0<                ( EXPECTING CHARS ? )
7     IF 7F #B 0 AND                       ( KILL PARITY BIT )
8         PTR U) W XCHG 0 W ) MOV B
9         W INC PTR U) W XCHG              ( STORE CHAR )
10        CNT U) INC CTR U) INC 0=         ( CTR = 0 ? )
11    IF WAKE # STATUS U) MOV              ( AWAKEN AWDS TASK )
12    THEN
13 .S
14
15

```

## 591 LIST

```

0 ( INTER-ASSEMBLY INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3     ELSE 1 #B UARTBFLG TEST  0= NOT      ( TEST UART LOOPBACK ? )
4         IF 7F #B 0 AND      ( KILL PARITY BIT )
5             0 CHARBUFFB MOV B      ( SAVE LOOPBACK CHAR )
6             0 #B UARTBFLG MOV      ( COMPLETE TEST )
7         THEN
8     THEN ireturn JMP
9
10 .S
11
12
13
14
15

```

## 592 LIST

```

0 ( AWDS INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL (INTR3) ( UART B SPECIAL RECEIVE COND. INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH
5     01 #B 0 MOV awds/csr OUT      ( POINT TO RR1 B )
6     awds/csr IN                    ( GET REC. ERROR COND. )
7     30 #B 0 MOV awds/csr OUT      ( RESET ERROR COND. )
8     ireturn JMP
9
10
11
12 .S
13
14
15

```

## 593 LIST

```

0 ( INTER-ASSEMBLY INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL (INTR4) ( UART A TBRE INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH      ( SAVE REGISTERS )
5     'COMM.TASK # U MOV
6     CTR U) 0 MOV 0 0 OR 0)          ( HAVE ANY CHARS TO XMIT ? )
7     IF PTR U) I XCHG LODS B PTR U) I XCHG
8         comm/data OUT CTR U) DEC
9         5 #B CTR U) CMP 0=          ( PASSED 2ND CHAR ? )
10    IF 1 #B UARTAFLG MOV            ( FLAG SAVE 1ST LOOPBACK CHAR )
11    THEN
12    ELSE WAKE # STATUS U) MOV        ( AWAKEN I/A TASK )
13        28 #B 0 MOV comm/csr OUT    ( RESET XMIT INTERRUPT )
14    THEN ireturn JMP
15 .S

```



## 597 LIST

```

0 ( INTER-ASSEMBLY INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3     THEN
4     ELSE 1 #B UARAF LG TEST 0= NOT ( TEST UART LOOPBACK ? )
5         IF 0 CHARBUFFA MOV B ( SAVE LOOPBACK CHAR )
6             0 #B UARAF LG MOV ( FLAG END OF TEST )
7     THEN
8     THEN ireturn JMP
9
10 .S
11
12
13
14
15

```

## 598 LIST

```

0 ( INTER-ASSEMBLY INTERRUPT HANDLER                23/NOV/86 )
1
2 HEX
3 LABEL (INTR7) ( UART A SPECIAL RECEIVE COND. INTERRUPT HANDLER )
4     0 PUSH 1 PUSH 2 PUSH U PUSH
5     01 #B 0 MOV comm/csr OUT ( POINT TO RR1 A )
6     comm/csr IN ( GET REC. ERROR COND. )
7     30 #B 0 MOV comm/csr OUT ( RESET ERROR COND. )
8     ireturn JMP
9
10
11 .S
12
13
14
15

```

## 599 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

600 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

601 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

602 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

603 LIST

```
0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

604 LIST

```
0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

605 LIST

```
0 ( TASK USER VARIABLES INITIALIZATION          30/APR/85 )
1
2 HEX
3 | CREATE OPERATOR 'OPERATOR , EA2E , 'COMM.TASK , ZERO ,
4   0 , 'OPERATOR 61 - , OA ,
5
6 | CREATE COMM.TASK 'COMM.TASK , EA2E , 'WDOG , ZERO ,
7   0 , 'COMM.TASK 61 - , OA , ' (TYPE) , ' (EXPECT) ,
8   ' (CR) , ' (PAGE) , 0 , 0 , 0 ,
9
10 | CREATE WDOG 'WDOG , EA2E , 'AWDS.TASK , ZERO ,
11   0 , 'WDOG 61 - , OA ,
12
13 .S
14
15
```

## 606 LIST

```

0 ( TASK USER VARIABLES INITIALIZATION          30/APR/85 )
1
2 HEX
3 | CREATE AWDS.TASK 'AWDS.TASK , EA2E , 'PRINT.TASK , ZERO ,
4   0 , 'AWDS.TASK 61 - , 0A , ' (TYPE) , 0 , ' (CR) ,
5   ' (PAGE) , 0 , 0 , 0 ,
6
7 | CREATE PRINT.TASK 'PRINT.TASK , EA2E , 'DATA.PROC , ZERO ,
8   0 , 'PRINT.TASK 61 - , 0A , ' (TYPE) , 0 , ' (CR) ,
9   ' (PAGE) , 0 , 0 , 0 ,
10
11 | CREATE DATA.PROC 'DATA.PROC , EA2E , 'OPERATOR , ZERO ,
12   0 , 'DATA.PROC 61 - , 0A ,
13 .S
14
15

```

## 607 LIST

```

0 ( McCLELLAN ASSY SYSTEM INITIALIZATION )
1
2 HEX
3 : INIT.ASSEMBLY ( INIT FOR EITHER INDICATOR OR RECORDER ASSY )
4           FF LEDSTATE C!           ( TURN LEDS OFF )
5           CONFIG C@                 ( GET CONFIGURATION )
6           ACT.SENSOR?               ( GET ACTIVE SENSOR )
7           OVER 10 AND NOT           ( RECORDER ASSY ? )
8           IF 10 OR                  ( SET FOR NORM. DISP)
9           EF LEDSTATE C!           ( TURN ON LED 1 )
10          THEN SENSOR# C! 8 AND      ( MASTER CONFIG. ? )
11          IF 1                       ( FLAG POLL SENSOR )
12          ELSE 0                     ( FLAG EAVEDROP )
13          THEN CONTRLFLAG C! ;       ( SAVE FLAG )
14
15 .S

```

## 608 LIST

```

0 ( McCLELLAN IND/REC      ASSEMBLY CONFIGURATION ROUTINE  1/28/87)
1
2 HEX
3 CODE read.config ( -- read links to establish configuration)
4   600 # 1 MOV           ( wait for mode change)
5   BEGIN 50 #B 0 MOV fpi/csr OUT ( read fifo, auto increment)
6     1 W XCHG 08 # 1 MOV
7   BEGIN fpi/data IN           ( read thru regs)
8   LOOP 1 W XCHG E0 # 0 MOV fpi/csr OUT ( end interrupt )
9   LOOP 40 #B 0 MOV fpi/csr OUT ( set to read location 0)
10  fpi/data IN E0 #B 0 AND ( read data, clear d0-d4 )
11  3 # 1 MOV 0 RCL B V CS ( rotate d7 & 6 to d1 & 0)
12  IF 10 #B 0 OR           ( set bit 5 if indicator)
13  THEN 0 2 MOV B          ( save results in 2)
14  46 #B 0 MOV fpi/csr OUT ( set to read config sw)
15  fpi/data IN 1 #B 0 TEST 0= NOT ( mbr0 set ?)

```

## 609 LIST

```

0 ( McCLELLAN IND/REC      ASSEMBLY CONFIGURATION ROUTINE  1/28/87)
1
2 HEX
3
4     IF 4 #B 2 OR
5     THEN 47 #B 0 MOV  fpi/csr OUT      ( set bit 2, REGULAR mode )
6         fpi/data IN  1 #B 0 TEST 0= NOT ( read mbr1 )
7         IF 8 #B 2 OR
8         THEN 0A #B 0 MOV  fpi/csr OUT   ( mbr1 set ? )
9             80 #B 0 MOV  fpi/csr OUT   ( set bit 3, MASTER)
10            29 #B 0 MOV  fpi/csr OUT   ( set to keybd mode)
11            mcs/reg IN
12            80 #B 0 TEST 0= NOT       ( clear irq )
13            IF 20 #B 2 OR
14            THEN 2 CONFIG MOV B      ( set divisor )
15            NEXT                    ( read mcs/reg reg. )
                                     ( 2 minute ave. ? )
                                     ( set bit 5 )
                                     ( save in CONFIG )
15 .S

```

## 610 LIST

```

0 ( McCLELLAN IND/REC      8274 USART INITIALIZATION  10/21/86)
1
2 HEX
3 CODE comm.init ( intialize inter-assembly usart )
4     18 #B 0 MOV  comm/csr OUT  comm/csr OUT ( reset twice)
5     04 #B 0 MOV  comm/csr OUT      ( point to wr4 ch. A)
6     44 #B 0 MOV  comm/csr OUT      ( set 16X 8-bits no parity)
7     01 #B 0 MOV  comm/csr OUT      ( point to wr1)
8     1F #B 0 MOV  comm/csr OUT      ( enable intrs )
9     02 #B 0 MOV  comm/csr OUT      ( point to wr2)
10    B0 #B 0 MOV  comm/csr OUT      ( 88 mode, var vector)
11    03 #B 0 MOV  comm/csr OUT      ( point to wr3 )
12    C1 #B 0 MOV  comm/csr OUT      ( Rx 8 bits, Rx enable )
13    05 #B 0 MOV  comm/csr OUT      ( point to wr5)
14    68 #B 0 MOV  comm/csr OUT      ( Tx 8-bits, Tx enable)
15 .S

```

## 611 LIST

```

0 ( McCLELLAN IND/REC      8274 USART INITIALIZATION  10/21/86)
1
2 HEX
3     00 # 0 MOV comm/csr OUT  10 # 0 MOV
4     comm/csr OUT  comm/csr OUT ( reset external status)
5     NEXT
6
7
8
9
10
11
12 .S
13
14
15

```

## 612 LIST

```

0 ( McCLELLAN IND/REC 8274 USART INITIALIZATION 10/21/86)
1
2 HEX
3 CODE awds.init ( initialize awds usart )
4     18 #B 0 MOV awds/csr OUT awds/csr OUT ( reset ch B )
5     04 #B 0 MOV awds/csr OUT ( point to wr4 ch. B)
6     47 #B 0 MOV awds/csr OUT ( set for 16X 8-bits, even par)
7     01 #B 0 MOV awds/csr OUT ( point to wr1)
8     1F #B 0 MOV awds/csr OUT ( enable intrs )
9     02 #B 0 MOV awds/csr OUT ( point to wr2)
10    08 #B 0 MOV awds/csr OUT ( set vector offset)
11    03 #B 0 MOV awds/csr OUT ( point to wr3 )
12    61 #B 0 MOV awds/csr OUT ( Rx 7 bits, auto enable )
13    05 #B 0 MOV awds/csr OUT ( point to wr5)
14    2A #B 0 MOV awds/csr OUT ( Tx 7-bits, Tx enable, RTS )
15 .S

```

## 613 LIST

```

0 ( McCLELLAN IND/REC 8274 USART INITIALIZATION 10/21/86)
1
2 HEX
3     00 #B 0 MOV awds/csr OUT ( point to wr0)
4     10 #B 0 MOV awds/csr OUT ( reset status vector)
5     awds/csr OUT ( reset status vector)
6     NEXT
7
8
9
10
11
12 .S
13
14
15

```

## 614 LIST

```

0 ( McCLELLAN IND/REC DEVEL DEVICE INITIALIZATION 8/22/86)
1
2 HEX
3 CODE init.fpi ( -- setup the 8279 fpi)
4     0C #B 0 MOV fpi/csr OUT ( setup as sensor matrix mode )
5     DF #B 0 MOV fpi/csr OUT ( reset the 8279)
6     DF #B 0 MOV fpi/csr OUT ( reset the 8279)
7     E0 #B 0 MOV fpi/csr OUT ( non-special error mode)
8     29 #B 0 MOV fpi/csr OUT ( setup divisor as 9)
9     NEXT
10
11 .S
12
13
14
15

```

## 615 LIST

```

0 ( McCLELLAN IND/REC DEVEL  DEVICE INITIALIZATION   8/22/86)
1
2 HEX
3 CODE init.rtc ( -- setup the 7170 rtc)
4     OC #B 0 MOV  clock/csr OUT    ( disable intr. & clock)
5     00 #B 0 MOV  clock OUT        ( clear 100drths reg )
6     1C #B 0 MOV  clock/intr OUT   ( setup 0.1 sec interrupt)
7     clock/csr IN                   ( clear any interrupts )
8     NEXT
9
10 .S
11
12
13
14
15

```

## 616 LIST

```

0 (  INTERRUPT VECTORS INITIALIZATION                30/APR/85 )
1
2 HEX
3 | CREATE INTERRUPTS  <INTR0> , ZERO , <INTR1> , ZERO ,
4                       <INTR2> , ZERO , <INTR3> , ZERO ,
5                       <INTR4> , ZERO , <INTR5> , ZERO ,
6                       <INTR6> , ZERO , <INTR7> , ZERO ,
7
8
9 .S
10
11
12
13
14
15

```

## 617 LIST

```

0 (  INTERRUPT VECTORS INITIALIZATION                30/APR/85 )
1 HEX
2
3
4 .S
5
6
7
8
9
10
11
12
13
14
15

```

## 618 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 619 LIST

```

0 ( INITIALIZE I/O HARDWARE                               30/APR/85 )
1
2 HEX
3 CODE start.int
4     1C #B 0 MOV clock/csr OUT    ( start clock & enable intr)
5     STI NEXT
6
7 : INIT.RAM ( CLEARS SYSTEM RAM )
8     [ THERE ] LITERAL RAMSTART - ( RAM BYTE USAGE )
9     2/ RAMSTART 2DUP
10    OVER ERASE hit.dog           ( CLEAR 1ST HALF )
11    + SWAP ERASE hit.dog ;       ( CLEAR LAST HALF )
12
13 .S
14
15

```

## 620 LIST

```

0 ( McCLELLAN  INITIALIZATION ROUTINES                   1/9/87 )
1
2 : INIT.HARD ( INITIALIZES SYSTEM HARDWARE CONFIGURATION )
3     init.fpi  init.rtc  comm.init
4     awds.init 255 led.hdlr ; ( TURN LEDs & ALARM OFF )
5
6 : ILL.SEN.DATA ( SETUPS DATABASE FOR UNDEFINED SENSORS )
7     3 ( MAX 4 ) ?.#SENSORS - ?DUP ( ANY UNDEFINED SEN ?)
8     IF 4 DUP ROT -
9     DO I RESULTBUF 32 SET        ( STUFF 555 W/ MSB SET TO )
10    DO 3323 I ! 2 +LOOP          ( INDICATE INVALID SENSOR )
11    LOOP hit.dog
12    THEN ;                       .S
13
14
15

```

## 621 LIST

```

0 ( McCLELLAN IND/REC TASK ACTIVATION ROUTINE      8/8/86)
1
2 : 1TASK  COMM.TASK ACTIVATE INTER.ASSY  ;
3 : 2TASK  AWDS.TASK ACTIVATE AWDS      ;
4 : 3TASK  WDOG ACTIVATE TEST.SYSTEM    ;
5 : 4TASK  DATA.PROC ACTIVATE PROCESS.DATA ;
6 : 5TASK  PRINT.TASK ACTIVATE PRINTER  ;
7
8 : TASKS  1TASK 2TASK 3TASK 4TASK      ( START TASKS )
9          CONFIG C@ 16 AND 0=          ( RECORDER ASSY ? )
10         IF 5TASK                      ( START PRINTER TASK )
11         THEN ;
12
13 .S
14
15

```

## 622 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : SUM.QUE ( a -- d  ADD CONTENTS OF QUE AT a AS DOUBLE)
3          0 0 ROT 240 SET      ( SETUP LOOP)
4          DO I @ M+ 2
5          +LOOP ;              ( LEAVE RESULTS ON STACK)
6
7 : BAD.SUM ( n -- RESTART 10 MIN. DATABASE OF SENSOR n)
8          DUP DUP              ( NEED n,n,n, -- )
9          INIT.QUES            ( INITIALIZE QUES )
10         INIT.SUMS            ( INITIALIZE SUMS )
11         FLAG.DATA ;          ( CLEAR PASS COUNT )
12 .S
13
14
15

```

## 623 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : CHECK.CHECKSUMS ( n -- f  VALIDATES CHECKSUMS )
3          )R I SPEEDQ SUM.QUE ( SUM SPEEDQ)
4          I SPEEDCK 2@ D=     ( CHECK STORED SUM)
5          I ANGLEQ SUM.QUE    ( SUM ANGLEQ)
6          I ANGLECK 2@ D=     ( CHECK STORED SUM)
7          I SUM.SUMS          ( CHECK RUNNING SUMS)
8          R) SUMCK 2@ D=
9          AND AND NOT ;      ( ANY NOT = ?)
10
11 .S
12
13
14
15

```

## 624 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : INIT.VARS ( INITIALIZES SYSTEM VARIABLES )
3           5 DUP SSECCTR C!      500MSCTR C!
4           1 DISPFLAG C!        0 STATDISFLAG C!
5           0 1MINCTR C!         0 CLOCKFLAG C!
6           0 AWDSFLAG C!        0 UARTBFLG C!
7           0 PRINTFLAG C!       0 FAILCTR C!
8           0 POLLFLAG C!        60 TIMEOUT C!
9           0 XMITFLAG C!        0 UARAFGL C!
10          0 CDFLAG C!          0 RESPFLAG C!
11          12345 POWERFLAG ! ;           ( FLAG 1ST STARTUP )
12
13 .S
14
15

```

## 625 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : VALIDATE.DATABASE ( TEST FLAGS TO CHECK FOR COMPLETE PROCESSG)
3           ?.#SENSORS 1+ 0      ( SET LOOP )
4           DO I CHECK.CHECKSUMS ( TEST CKECKSUMS)
5           IF I BAD.SUM         ( IF BAD, RESET SENSOR )
6           THEN I VALID C@ ?DUP ( PROCESSING DATA )
7           IF 16 =              ( FINISHED AVERAGES )
8           IF I WIND.DATA       ( COMPLETE PROCESSING )
9           ELSE I FLAG.DATA     ( REBUILD DATABASE )
10          THEN
11          THEN PAUSE
12          LOOP ;
13 .S
14
15

```

## 626 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : WARM.START ( PERFORM WARM STARTUP PROCEDURE )
3           ?.#SENSORS 1+ 0      ( GET SENSOR # )
4           DO I CHECK.CHECKSUMS ( TEST CHECK SUMS)
5           IF I BAD.SUM         ( IF BAD, RESTART)
6           THEN
7           I FLAG.DATA hit.dog  ( FLAG INVALID DATABASE )
8           LOOP ;
9
10
11
12 .S
13
14
15

```

## 627 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : HOT.COLD? ( d d -- DETERMINE TYPE OF RESTART )
3           D- 2DUP 61 0 D(      ( TIME DIFERENCE ( 60 SECONDS ?)
4           IF 6 0 D( NOT      ( GREATER THEN 5 SECONDS ?)
5           IF WARM.START      ( FLAG INVALID DATABASE )
6           ELSE VALIDATE.DATABASE ( CK IF PROCESS FINISH )
7           THEN
8           ELSE 2DROP INIT.RAM      ( INIT. SYSTEM DATABASE )
9           THEN ;
10
11
12 .S
13
14
15

```

## 628 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 .S
3
4
5
6
7
8
9
10
11
12
13
14
15

```

## 629 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : CK.TIME? ( COMPARES RTC TIME W/ TIME STAMP )
3           HOUR @DATE.TIME HR.SEC      ( CONVERT TIME TO SECS)
4           RAMHOUR @DATE.TIME HR.SEC   ( CONVERT STAMP TO SEC)
5           HOT.COLD? ;                  ( DETER. TYPE OF RESTART )
6
7 : CK.TIME2? ( COMPARES RTC TIME W/ TIME STAMP )
8           HOUR @DATE.TIME HR.SEC      ( CONVERT TIME TO SECS)
9           20864 1 D+                    ( ADD 1 DAY IN SECS )
10          RAMHOUR @DATE.TIME HR.SEC   ( CONVERT STAMP TO SEC)
11          HOT.COLD? ;                  ( DETER. TYPE OF RESTART )
12
13
14 .S
15

```

## 630 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      8/8/86)
1
2 : TEST.TIME ( COMPARE CLOCK TIME WITH TIME STAMP FOR RESTART )
3     MOD.DAY                                ( GET DATE IN MODULUS DAY )
4     RAMMONTH @DATE.TIME MOD.ANYDAY ( GET MOD. TIMESTAMP )
5     - DUP 0=                                ( SAME DAY ? )
6     IF DROP CK.TIME?                        ( COMPARE TIME )
7     ELSE 1 =                                ( OFF BY 1 DAY ? )
8         IF RAMHOUR clock) 23 =             ( TIMESTAMP = 23 HOUR & )
9         HOUR clock) 0= AND                 ( TIME = 0 HOUR ? )
10        IF CK.TIME2?                       ( COMPARE TIME )
11        ELSE INIT.RAM                      ( NEW START-UP )
12        THEN
13        ELSE INIT.RAM                      ( NEW START-UP )
14        THEN
15    THEN ;                                .S

```

## 631 LIST

```

0 ( INDICATOR/RECORDER START-UP ROUTINE      4/8/88)
1
2 : START-UP ( INDICATOR/RECORDER SYSTEM STARTUP ROUTINE )
3     hit.dog INIT.HARD hit.dog             ( INITIALIZE HARDWARE )
4     POWERFLAG @ 12345 =                   ( WARM START ? )
5     IF TEST.TIME                          ( COMPARE TIME & TIME STAMP )
6     ELSE INIT.RAM                         ( PERFORM COLD START )
7     THEN INIT.VARS                        ( INIT. SYSTEM VARIABLES )
8         read.config INIT.ASSEMBLY ( SET UP CONFIGURATION )
9         ILL.SEN.DATA hit.dog
10        TASKS start.int DISP.DATA ;
11
12 .S
13
14
15

```

## 632 LIST

```

0 ( INTENTIONALLY BLANK SCREEN )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

633 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

634 LIST

0 ( INTENTIONALLY BLANK SCREEN )  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

635 LIST

0 ( SYSTEM POWER-UP ROUTINE 30/APR/85 )  
1  
2 | : START [ RECOVER ]  
3 hit.dog  
4 INTERRUPTS [ ORAM 32 + ] LITERAL 32 MOVE  
5 0 2 ! [ ' DIVERR ] LITERAL 0 ! ( SET VECT 0 DIVERR )  
6 ( SETUP RAM )  
7 OPERATOR 2+ OPERATOR STATUS HIS DUP 60 ERASE 12 MOVE  
8 COMM.TASK 2+ COMM.TASK STATUS HIS DUP 60 ERASE 28 MOVE  
9 AWDS.TASK 2+ AWDS.TASK STATUS HIS DUP 60 ERASE 28 MOVE  
10 WDOG 2+ WDOG STATUS HIS DUP 60 ERASE 12 MOVE  
11 DATA.PROC 2+ DATA.PROC STATUS HIS DUP 60 ERASE 12 MOVE  
12 PRINT.TASK 2+ PRINT.TASK STATUS HIS DUP 60 ERASE 28 MOVE  
13 START-UP ;  
14  
15 .S

636 LIST

```
0 ( SYSTEM POWER-UP ROUTINE                               30/APR/85 )
1
2 HEX
3 LABEL POWER-UP 0 # 0 MOV
4                0 ES LSG  0 SS LSG  0 DS LSG
5                ' START 2- # I MOV  OPERATOR U MOV  U R MOV
6                OPERATOR OA + S MOV  NEXT
7
8 HERE  /PROM OF - ORG EA C, POWER-UP , ZERO ,
9      ORG
10
11 .S
12
13
14
15
```

637 LIST

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

638 LIST

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

