AL-TR-1992-0155

AD-A260 830

A R M S T R O N G   L A B O R A T O R Y

DEVELOPMENT OF A FLIGHT INSTRUMENT PACKAGE

Dan D. Fulgham
John L. Orr
Brian Mikiten

DTIC
ELECTE
FEB 17 1993
S E D

MacAulay Brown, Incorporated
3915 Germany Lane
Dayton, OH 45431

Southeastern Center for Electrical Engineering Education
1101 Massachusetts Avenue
St. Cloud, FL 34769

CREW SYSTEMS DIRECTORATE
2504 D Drive, Suite 1
Brooks Air Force Base, TX 78235-5104

December 1992

Final Technical Report for Period 16 November 1987 - 30 April 1991

93 2 16 045
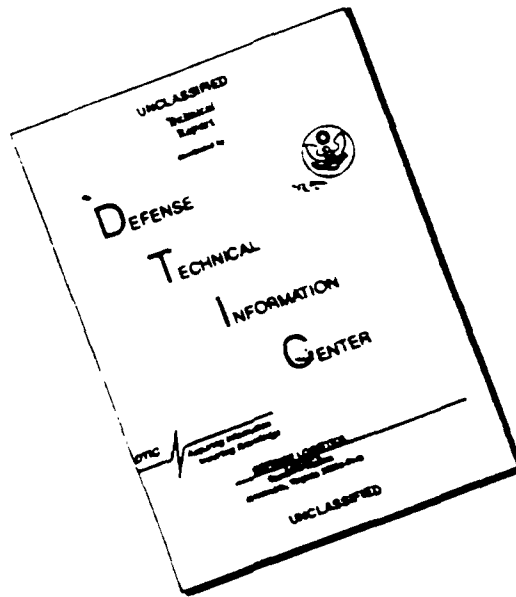
93-02959

AIR FORCE MATERIEL COMMAND
BROOKS AIR FORCE BASE, TEXAS

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

December 1992   Final - 16 November 1987 – 30 April 1991

C - F33615-87-C-0534
      (Task 0002)
Development of a Flight Instrument Package   C - F33615-87-D-0609
      (Task 0014)
PE - 61101F, 62202F
Dan D. Fulgham   PR - 7930
John L. Orr   TA - 14
Brian Mikiten   WU - 8E

MacAulay Brown, Inc.   Southeastern Center for Electrical
3915 Germany Lane   Engineering Education (SCEEE)   SwRI Project No. 12-2301
Dayton, OH 45431   1101 Massachusetts Avenue   Subcontract No. 1107-02-
      St. Cloud, FL 34769   08SWA

Armstrong Laboratory
Crew Systems Directorate
2504 D Drive, Suite 1
Brooks Air Force Base, TX 78235-5104   AL-TR-1992-0155

Armstrong Laboratory Technical Monitor: Dr. Kent K. Gillingham, (210) 536-3521

Approved for public release; distribution is unlimited.

Subcontractor (Southwest Research Institute) describes the hardware and software comprising the Flight Instrument Package (FIP), a collection of transducers and electronic components that measure primary flight motion and position parameters and generate digital data representing those parameters. Pitch angle, bank angle, altitude, vertical velocity, airspeed, heading, and angle of attack are the main quantities digitized and relayed to a data port for processing into various displays of aircraft state. The FIP developed as part of this task was used in a Beech Queen Air aircraft to drive the Acoustic Orientation Instrument, which provides the pilot with an auditory display of aircraft bank, airspeed, vertical velocity, and other parameters as necessary.

Acoustic orientation   400
Aural displays
Flight instrumentation

Unclassified   Unclassified   Unclassified   UL

i

NOTICES

This technical report is published as received and has not been edited by the technical editing staff of the Armstrong Laboratory.

This report has been reviewed and is approved for publication.

KENT K. GILLINGHAM, M.D., Ph.D.
Project Scientist

RONALD C. HILL, Lt Col, USAF, BSC
Chief, Flight Motion Effects Branch

RICHARD L. MILLER, Ph.D.
Chief, Crew Technology Division

DTIC QUALITY INSPECTED 3

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

TABLE OF CONTENTS

List of Figures:

Appendices:

INTRODUCTION

The Flight Instrument Package (FIP) is designed to translate inflight motion to digital and analog signals that can be used by the Acoustic Orientation Instrument or AOI. Information on the AOI can be found in the report titled Inflight Evaluation of an Acoustic Orientation Instrument Final Report, March 1990. The sensor and data processing package provides real-time data corresponding to aircraft speed, altitude, vertical velocity, bank angle, pitch angle, heading, and angle of attack. Both digital RS-232 data and processed analog signals are available to the user. A Beech Queen Air served as the test aircraft for the development of the system. During testing of the individual transducers, the pilots were asked to accomplish a series of repetitive maneuvers that tested specific attributes of each device. The data taken during these test flights was correlated with verbal reports and the accuracy of the transducer was determined during post processing of the flight data.

HARDWARE

The design of the FIP includes four major subsections (Figure 1: ᶜ   m Overview). The first component, sensors, allows the system to ι.ιᴜnitor movement, air pressures, and angular changes while in flight. The second, sensor electronics, provides the necessary translation, amplification, and filtration to the raw sensor signals. The third, a computer system, provides the necessary interface and data processing capabilities required to both analyze the sensor data and provide meaningful output to the AOI. Finally, a power source and convertor provide power to both the FIP and the AOI.

The FIP contains several transducers which translate physical parameters to electrical to be processed by the microcomputer system. Because of the sensors utilized in this design, in most cases, the response rate and accuracy of the FIP are far greater than that of the aircraft instruments.

A total of nine sensor parameters are used by the computer system to characterize flight performance. These are:

Heading (X, Y, and Z)
Airspeed
Altitude
Vertical Velocity (calculated)
Bank Angle
Pitch Angle
Heading Deviation
Altitude Deviation
Angle of Attack

In several cases, the signals for the sensors are used in combination with each other to calculate a parameter. Two examples, heading and vertical velocity, are discussed. Raw heading is first calculated using the X, Y, Z magnetometer signals and a simple transformation algorithm. It is then

SENSORS

Rate Gyro

Differential Pressure Transducer

Static Pressure Transducer

3 Axis Magnetometer

AOA VANE

PITOT
STATIC

CABIN
PRESSURE

SIGNAL CONDITIONING

R → V and Gain

Filters

Buffers

12 bit A/D

68010/68881 w/
Mach II ROMS

FORTH Software

COM1

COM2

Data Logger

AOI 2

PILOT

TEST CONNECTOR

BATTERY PACKAGE

FIGURE 1 - SYSTEM OVERVIEW

translated to an upright condition by subtracting the roll and pitch angles shown by the gyroscope. This re-orients the gyro to a position that is horizontal to the Earth's axis or the null position. Vertical velocity is derived using a change in altitude over time. Altitude data is provided using a pressure transducer and the time from a real time clock.

Sensor Overview

FIP sensors provide a conversion of physical parameters to an electrical signal that can then be interpreted by the FIP computer system. (Figure 2: Sensors). The FIP uses a total of five sensors in order to provide this capability. Each is listed and described below.

Airspeed

Airspeed is a function of the differential pitot and static pressures in the aircraft's pitot-static system. The co-pilot's pitot-static system in the Beech Queen Air has been modified to allow monitoring by the FIP. In the case of airspeed, the pitot side of the system is led to the high pressure end of the transducer. The low end of the transducer is connected to the static side of the aircraft and the resulting differential pressure provides an accurate measurement of airspeed over varied temperature and humidity conditions. If the flight pack is flown above 12,500 feet or at temperature ranges beyond standard commercial ranges, software compensation and additional hardware capable of monitoring outside temperature and humidity will be required.

The transducer provides a 10 mv per PSI output and is specified with a maximum sensitivity of 5 PSI. The electronics in the FIP provide a 50 k load, 5 volt negative offset, and a gain of 1000 to give a total output voltage of ±5 volts.
During the initial testing of the system, airspeed was calculated using the formula:

$$P = 1/2\ p\ ^*\ (V)^2\ (\ 1\ +\ (1/4)[(V^2)/(a0^2)])$$

where:

P = differential pressure reading
p = Air density at sea level
V = Airspeed
a0 = Speed of sound at sea level
V = velocity.

Differences in humidity, initial air temperature, and relative angle of the sensors to the plane cause anomalies in the data which are compensated for by using a look-up table mapped to the PSI values (Appendix A).

Altitude

FIGURE 2 - SENSORS

Altitude is derived using a 15 PSI pressure transducer. The transducer provides 100 mv full scale sensitivity using the 10 volt reference included in the FIP circuitry. This signal decreases as the aircraft increases altitude because air pressure decreases as altitude increases. The static readings from the transducer are multiplied by a gain of 100 to provide full scale output. The signal is offset by a negative 5 volts to provide better resolution to the A/D board.

The minimum theoretical altitude of 15 PSI occurs below sea level and produces an A/D count of 4096. Since this is rarely possible in normal flight, the range of 0-15 PSI was deemed acceptable for the sensor range. A maximum altitude of 13,000 feet above mean sea level was used in this application. A reading of 2.39 PSI at 13,000 feet yields a count of 653, the maximum altitude to be attained in the Beech Queen Air.

A lookup table is utilized because of the complex equations used to determine altitude. Appendix B shows the altitude vs. PSI data used in the FIP. During the course of development, several equations fit to these data resulted in unacceptable accuracy. The lookup table is implemented in the code and shown in Appendix C.

During the final testing of the system, an initial altitude offset was entered while downloading the code to the FIP as the airplane was on the field before takeoff. This helped to null the effect of temperature differences found in the summer at the various altitudes. (see System Software below)

Vertical Velocity

Vertical velocity is a calculated value based on a change in altitude over time. The system uses a time base of 15 seconds for the calculation. The current value for altitude at time zero is subtracted from the altitude value at time zero plus 15 seconds. Provisions are made to alter the time period used for more rapid updates. This signal is not directly derived with a transducer, therefore, the signals are not directly related to A/D counts. In order to allow for both climbing and diving, the A/D counts above 2048 are used to show a climb and the numbers below 2048 represent a dive. An A/D count of zero represents the maximum dive value and the A/D count of 4096 represents the maximum climb. Because these numbers are set during downloading, the computer source code (Appendix C) should be consulted for the current settings.

Angle of Attack

This signal was originally provided from a synchro based angle of attack (AOA) vane mounted on the side of the aircraft. Initial circuitry consisted of a synchro to linear convertor, power convertor, and a set of buffer amplifiers. Incompatibilities in the system prevented the units from working properly and

the AOA system was not implemented using the synchro outputs. Instead, the AOA vane was modified by removing a single non-working synchro and replacing it with a potentiometer. While the system remains untested, it is expected to provide accurate results with only minor circuit modifications.

Vertical Gyroscope

A vertical gyroscope from Humphrey, Inc. (Appendix D) was used to determine bank and pitch angle. The calibration data (Appendix D) provides reasonable accuracies and allows a straight line interpolation of the data. Because of the mechanisms used within the gyro, several precautions are necessary. First, the gyro should be shielded from any mechanical shock. A special vibration absorbing mounting material is used to limit the mid and high frequency vibrations produced by the aircraft during flight. Because of this, bolts or other securing devices are not required and, in fact, would simply transmit the harmful vibrations if installed. Secondly, the gyro should not be turned on unless the FIP is being used. The life of the bearings is limited and unnecessary use simply shortens the time between servicing. Contact the manufacturer if servicing is required.

The implementation of both bank and pitch is shown below. Care has been taken to limit the current provided at the gyro electrical contacts. Because of this current limiting, any modifications to the gyro portion of the FIP should be done in accordance with the gyro literature.

Bank Angle

Bank angle is derived using a vertical gyroscope. The gyro provides a stabilized source around which a mechanically coupled potentiometer moves. As the relative position of the gyro changes, a resistance change proportional to the change in position occurs. The maximum bank angle to the left produces an A/D count of zero, while a maximum bank angle to the right produces a count of 4096. The null position for the sensor produces 2048 counts. Typical values during test flights were between 681 and 3415 A/D counts corresponding to +20 and -20 degrees of roll.

Pitch Angle

Pitch angle is derived in a manner similar to that of bank angle. The maximum pitch up occurs at +60 degrees or zero A/D counts. The maximum pitch down occurs at -60 degrees or 4096 A/D counts. Null pitch, or zero degrees is represented by 2048 counts. Typical values during test flights were +25 to - 25 degrees of pitch corresponding to 1194 - 2901 A/D counts. Because of the low delta value and the obvious resolution errors that could occur, the gain on the op-amps may be doubled or tripled if the effective range is not required.

Three - Axis Magnetometer

The three-axis magnetometer, manufactured by Dowty Defense & Air Systems Limited, U.K., provides three axes of relational data. Each of the integral coils provides one of three angular measurements of the Earth's magnetic lines of flux. Output is provided as a ± 3 VDC signal proportional to the position of the sensor angle to the Earth's flux lines. As an example, when a sensor is parallel to the lines of flux, the resulting output is 0 VDC. If the sensor is perpendicular, maximum saturation occurs and maximum voltage is obtained. In this application, the three-axis strapdown magnetometer is used with the vertical gyro to determine heading. The magnetometer provides a north reference when nulled to a level condition using the angular offsets provided by the pitch and roll gyro. The dip angle is measured by the Z axis and the output data is read using the pitch gyro to null the dip angle. The nulled position provides an X and Y angular output. The arctangent of the X and Y provides the aircraft heading. A dip angle of 62 degrees is used but can be changed within the software if the FIP is used at a location other than the state of Texas.

Axis Data

X - Bank (wing to wing)
Y - Pitch
Z - Dip angle

Because of the dynamics of flight, the assigned coils are not fixed but instead flexible based on the mounting of the gyro. System repeatability, accuracy, sensitivity, and frequency response are shown in Appendix E.

The magnetometer has been mounted using a standard strip adhesive. Non-ferrous mounting of the device and the removal of moving ferrous based objects near the device during flight are important considerations that will insure accurate readings.

Heading Deviation

These data are derived using the difference between the calculated heading from the three-axis magnetometer and the setting shown on the front panel display. The compass headings are divided into the range of the A/D to calculate the values given by the FIP as shown below.

Ex: 4096/360 = counts / degree of error = 11 counts per degree of error

A potentiometer is used to set the preferred heading. The high terminal of the pot is provided with a 10V reference source. The low terminal of the pot is tied to ground. The wiper provides a user selectable voltage that is then buffered and read by the A/D board and displayed using a digital panel meter (see the section entitled Display Board below).

FIP ELECTRONICS

Signal Conditioning Board

The internal electronics in the FIP are located on three printed circuit boards. The main board, affixed to the base of the package, contains the sensor signal conditioning electronics and buffer electronics. The board mounted on the front of the FIP contains the battery pack power regulation and filter electronics used to convert the battery supplies to voltages used within the FIP and by the AOI. The final board provides the stabilized reference sources for the potentiometers used to set both desired heading and altitude.

The signal conditioning system electronics (Appendix F) are designed to work with either bridge-based or direct output sensors (Figure 3 Signal Conditioning Electronics). If a bridge-based transducer is used, a complete bridge and bridge amplifier are required if not specified in the original design. Input amplifiers are either single ended or differential. Both the airspeed and altitude sensors require that the pre-amplification stage configuration be differential with gains of 200. In all other cases, input gains are minimal.

All sensors in the system were tested individually and found to have a high frequency component that was not part of the useful data set. To eliminate this, a standard input configuration was used. As shown in Figure 3, all sensors are buffered using either unity gain op-amps (OP-77) or a pre-amplifier configuration with a minimum gain as mentioned above. The next level of processing includes a low pass filter with a roll offset to approximately 10 Hz. The discrete implementation using an MC34084 provides a one chip filtration that increases the system accuracy while providing the following amplification stages with a clean low-level signal. Each signal is then buffered twice before routing to the external analog connector or the system VME A/D converter board. Gains on both buffers can be modified, but are set to provide the maximum resolution to the A/D system. If gain changes are made in the system, the program constants must also be changed in order to limit scaling errors. In some cases, a -5 volt offset is added to allow for greater voltage swings within the working ranges of the sensors.

Both the analog test connector and the A/D connector can be used to directly monitor individual sensor operation. In most cases, the analog connector is most appropriate for this purpose. Pin numbering for the analog connector is shown below. The pin numbers for the A/D connector can be derived using the data presented in the Mizar Analog Input Module manual (Appendix I) and the channel numbering data shown in the section entitled Data Processing Computer System.

The analog test connector provides the sensor signals on a 26 pin Berg connector on the face of the FIP chassis. These pins provide only raw, and in some cases, amplified signals to the user for general debugging and analysis. The schematic for this board shows the output stages to this connector. Note

FIGURE 3 - SIGNAL CONDITIONING ELECTRONICS

that the OP-77 op-amps are the only buffering provided for these signals. Because of this, care should be taken to limit the output current when driving any significant loads.

The A/D channels, the signals being read, the connector numbering and the signal names are shown below.

Signal Name	Channel #	Berg 50 Pin	Berg 26 Pin
Altitude	4	13	20
Airspeed	5	17	22
Pitch angle	6	19	24
Roll angle	7	23	26
Mag. X-axis	8	2	18
Mag. Y-axis	9	4	16
Mag. Z-axis	10	8	14

Note: All unused inputs have been tied to ground on the printed circuit board. This helps not only to limit the noise in the system but also to prevent errors in software development.

Pins grounded on the Mizar A/D board:
3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,49,50

Power Supply Board

Because the power supplies available in the aircraft were not appropriate for this system, a battery package (see below) and power convertor system were required. The power supply board (Appendix G) consists of two Mizar DC-DC convertors (Appendix G). These units provide 5 volt outputs from 12 volt sources. The FIP utilizes +24V, ±12V, and +5 V as shown below. The + and - 12V supplies are used directly from the battery sources and filtered as necessary to eliminate reflected noise. The FIP also requires a 24V supply for the vertical gyroscope. This is derived using two -12 V battery sources in series. The 5V supply was derived using the Mizar convertor. Because the AOI is also powered from this system, a secondary 5V Mizar convertor was required. The + and - 12V signals for the AOI are similarly filtered. Power requirements and the voltages used by each portion of the system are shown below.

AOI power requirements
AOI digital board	5V @ 300ma
AOI 68HC11 board	5V @ 100ma
Timer	5V @ 50ma
Communications board	5V @ 100ma
Audio Board	5V @ 50ma
	+12V@ 300ma
	-12V@ 300ma

White noise and filter board   5V @ 50ma
                                      -12V@ 150ma
                                      +12V@ 150ma

FIP power requirements

VME system

    SBC-2   5V @ 3.6A
                                      +12V @ 35ma
                                      -12V @ 35ma
    VME-750   5V @ 1.2A
                                      +12V @ 35ma
                                      -12V @ 35ma
    Co-processor   5V @ 150ma

    Mizar A/D   5V @ 1.2 A

The use of Mizar power convertor modules caused several grounding problems in the system. The first problem, the occurrence of noise above 1MHz was expected because of the switching characteristics of the convertors and is suppressed using extensive bypassing and liberal ferrite noise suppressing. In several instances, a ground potential in excess of .5V was observed with connected grounds. This was especially noticeable in the computer system. In these cases, a single point ground is utilized.

Fuses

Fuses located on the board have been rated for twice the normal power requirements of the system and, in certain cases, for twice the surge requirements of sensors. If a fuse is blown, careful inspection is required to determine the source of the problem. If the fault was mechanically induced, replace the fuse with an identically rated fuse. If the fault was electrically induced, the cause of the fault should be determined before the replacement fuse is introduced into the system. Replacement of a faulty fuse when an electrical failure has occurred will likely cause a further fault in the system.

Battery Package

An external battery package is used to power the FIP/AOI in flight. The batteries are mounted in a transportable black case. Structural foam has been added to prevent battery movement while transporting the case. A single AMP-type circular connector delivers the unregulated battery voltages to the FIP power converter board. A pin-out of the connector and the list of batteries is provided in Appendix J. Placement within the case is shown in Figure 4.

(1) 12V -> 5V

(2) 12V -> 5V

(3) 24V -> 24V

(4) +12 -> +12

(5) +12 -> -12

FIP battery 12V (1)

AOI battery 12V (2)

24V

GYRO

6V | 6V | 6V | 6V

(3)

Analog +12V (4)

Analog -12V (5)

FRONT

FIGURE 4 : BATTERY PLACEMENT

Under normal flight conditions, the FIP/AOI can be powered for at least 2.5 hours. Once the power has drained from the system, the individual batteries must be re-charged before another flight can occur. Several re-chargers are included with the system which require connection to an AC power outlet. In most cases, the batteries can be returned to an operation state in 4 to 6 hours. Because of the chemical properties of the batteries, it may be necessary to deep cycle the batteries at regular intervals. If this is done correctly, the batteries will not develop a memory, thus preventing full charging. A simple way to deep cycle the batteries is to leave the FIP/AOI power on while the system is connected. Unfortunately, this method reduces the life of the gyro and will produce unknown voltages at the op-amps and the DC-DC switching supplies. A safer method uses high wattage resistors placed across the terminals of the regulator or the power connector. A typical value for each 12V battery is a 24 ohm 10 watt resistor.

The Display Board

The display board contains two Acculex LCD-type displays, a pair of potentiometers used to set the heading and altitude fly-to marks, the system power switches, and the AOI power connector. The schematics and board layouts for this board are shown in Appendix H. Data sheets for the Acculex displays are in Appendix H.

This board allows the operator to selectively turn on and off the Gyro power, AOI power, and FIP system power. The FIP power is wired in series with both the AOI and Gyro power and therefore needs to be on for the other components to work. Upon power-up, the Altitude Set and Heading Set displays will show a number based on the current value of the 10-turn pots. Changing the pot position will in turn change the number shown on the display and the voltage applied to two channels of the A/D convertor (see below). To achieve greater stability and system accuracy, REF-01 (Appendix H) voltage references were used in the FIP. These sources provide the highly stable Vin signal for the potentiometer voltage divider configuration over varied temperatures and input voltages. The ribbon connector on the board carries both the Display Board and Signal Processing Board signals to the A/D convertor board. Because of this, care must be taken when re-assembling the system to ensure that the connectors are properly engaged. A complete list of parts is shown in Appendix K.

DATA PROCESSING COMPUTER SYSTEM

The computer system in the FIP is a VME-bus based 68010 processor, VME/750 accessory interface board, and a Mizar 8605 analog to digital convertor board. Data sheets and instruction manuals for the 68010 board are provided in Appendix L. The manual for the VME/750 board is shown in Appendix M. The manual for the Mizar 8605 board in shown in Appendix I.

SBC-2 68010 board

The VME Specialist's 68010 board is a VME compatible microprocessor with a 10-Mhz clock, 512k of dual port RAM, two RS-232 ports, one 16 bit timer/counter, seven levels of interrupts, and EEPROM start-up capabilities. The board is a standard 3U unit and occupies the first slot of the card cage.

VME/750

The VME/750 multi-function accessory board is provided with the SBC-2. The VME-based board supports two additional RS-232 ports, a 16 bit counter/timer, a real-time clock with battery backup, sockets for additional RAM or ROMS, and a 68881 math co-processor. The board is tied directly to the SBC-2 using board-to-board jumpers and does not utilize the VME backplane for either data or address signals. The manual for this board is provided in Appendix I.

MACH-2 FORTH/68000/VME

Mach-2 Forth from Palo Alto Shipping Company was provided in ROM to VME Specialists for this application. A high and low address ROM is located on the SBC-2 board and is configured as the start-up system. Full floating point support is provided in the code and access to the 68881 math co-processor is required for proper operations of the unit. Code listings are provided in Appendix C.

MIZAR 8605 A/D

The Mizar 8605 analog input module provides a 12 bit VME-based A/D convertor. The board allows either 16 single ended or eight differential analog inputs at user selectable voltage ranges. Signal amplification to 1000x is also provided on the board. In this application, the board is configured to allow 16 single-ended inputs from -10 to +10 volts DC. Multiplexed, 25us conversion provides data rates far in excess of those required by this application. Connector data, conversion tracking information, schematics, and a parts list are provided in Appendix I.

SYSTEM SOFTWARE

The FIP software is written to run on the VME Specialists SBC-2/750 combination board in conjunction with the Mizar 8605 A/D board. The entire system uses the VME standard bus hardware configuration and is housed in a Electronic Solutions five slot card cage. The program is written in the Mach-2 Forth language located in ROM on the VME Specialists VME-750 board. The Flight Pack system is described best by splitting it in three parts. The three parts, input, processing, and output, each work together to form the complete processing loop.

Input

Eleven of the 16 available A/D channels are used in this application for signal inputs. Each channel is median-filtered using a constant filter size set by a program constant. A data set for an input channel is acquired by collecting a fixed number of values set by the filter size. Next the values are stored into an array and a bubble sort performed. Then, the middle value in the array is used as the current data value. This method proves effective in filtering out undesirable fluctuations that could occur from sampling data in a hostile environment.

All A/D channels are converted by the Mizar 8605 A/D board. The converted data is in 12-bit binary values. Some channels will have varying voltages converted to 12-bits which could be -10V to +10V (0 to 4096), -10V to 0V (0 to 2048), and 0V to +10V (2048 to 4096). The software scales all possible voltages accordingly to provide the highest system accuracy. The following will describe the characteristics of each of the channels.

Channel 0: Altitude Set

This parameter is set using a potentiometer located on top of the FIP. The value read is used as a target altitude for the pilot. The readings are scaled from 0 to 12,000 and the voltage output is from 0V to +10V.

Channel 1: Heading Set

This is set through a resistor pot. The value read is used as a target heading for the pilot. The readings are scaled from 0 to 360, and the voltage output is from 0V to +10V.

Channel 2: Not used

Channel 3: Not used

Channel 4: Altitude

This signal is read from a pressure transducer ranging from 0 to 15 PSI. The voltage scale is -10V to 10V. Note that the output voltage decreases with altitude.

Channel 5: Air Speed

This is read from a pressure transducer ranging from 0 to 5 PSI. The voltage scale is -10V to 0V. As pressure increases, airspeed increases.

Channel 6: Pitch

This is read from the vertical gyroscope with a range of ±50 degrees. The voltage scale is 0V to 9.32V. For a 12-bit A/D, the values are from 2048 to 3957.

Channel 7: Roll

This is read from a gyroscope with a range of ±70 degrees. The voltage scale 0V to 9.15V. For a 12-bit A/D, the values are from 2048 to 3922.

Channel 8: Y
Channel 9: X
Channel 10: Z

These are read from the magnetometer. The magnetometer senses the relation of its position with the Earth's magnetic field. The voltage scale is ±3 volts for each channel. For a 12-bit A/D the values are from 1434 to 2662.

The X-axis is the roll position.
The Y-axis is the pitch position.
The Z-axis is the dip angle position.

Processing

The Flight Pack calculates seven variables using the data obtained from the input channels. In addition to the scaling of the values read from the input channels, the data is scaled again for use by the AOI system. All variables are 16-bit values.

Variable 1: Air Speed

The raw PSI for air speed is taken from channel 5 and put through a lookup table. The lookup table has a knots-to-PSI correlation. By using an index to the knots and getting the closest matching PSI value, the index is used as the realized airspeed in knots. The realized knots are then scaled to 0 to 4096 for a range of 0 to 330 knots.

Variable 2:   Angle of Attack

    Not implemented

Variable 3:   Vertical Velocity

    Using a constant interval of five seconds, two separate readings of altitude are used to find the difference of altitude over a fixed period of time. The scaling of the final value is from 0 to 4096 with 2048 being no vertical velocity, 0 for a 2048 feet per second or greater dive, and 4096 for a 2048 feet per second or greater climb.

Variable 4:   Heading Deviation

    Heading value is calculated using the X, Y, Z, and pitch and roll variables. The deviation of the calculated heading from the heading set value in channel 1 is the final value. A scale of 0 to 360 is set for heading deviation.

Variable 5:   Roll

    Roll is read from channel 7 and is scaled to 0 to 4096 with 2048 being 0 degrees, 0 for 70 degrees left, and 4096 for 70 degrees right.

Variable 6:   Pitch

    Pitch is read from channel 6 and is scaled to 0 to 4096 with 2048 being 0 degrees, 0 for a 50 degree dive angle, and 4096 for a 50 degree climb angle.

Variable 7:   Altitude Deviation

    The PSI value is read from channel 4 and is put through a lookup table that is a best fit model of a table obtained from the standard atmospheric pressure table. The altitude is used with the altitude set value from channel 0 to find the difference or the altitude deviation. The difference is scaled from 0 to 4096 with 2048 being zero feet of deviation, 0 for -2048 feet or less deviation, and 4096 for +2048 feet or greater of deviation.

    During processing of the input channels, the FIP data can be requested for transmission at any time by the AOI system. To handle this request, a multi-tasking environment has been implemented on the VME Specialist system board. Processing of the input channels is performed in the foreground while communications are performed in the background.

    The background process checks the serial port number for the AOI transmit data request. In order to prevent the two processes from accessing the memory locations of the FIP data simultaneously, two data buffers are setup. One buffer is set as the current buffer for data processing and the other buffer is the completed, processed data ready for transmittal. The two buffers are switched between these assignments to prevent sending incomplete data to the

FIP software. When the transmit data request arrives, the buffer containing processed data is sent to the AOI system.

Output

The seven variables that have been calculated are transmitted upon receipt of a STX or Start TeXt (ASCII 2) character from the AOI system. Since the seven values are 16-bit values, and the RS-232 lines only handle 8-bit values, the 16-bit values are converted into 2 8-bit characters which are used for transmission over RS-232 lines. The sum of the byte representation of the seven values is used as a checksum to assure proper transmission. The transmission packet format of the 7 variables are as follows:

```
SOH  (ASCII 1 - Start Of Header)
AIRSPEED.HI                    AIRSPEED.LO
ANGLE OF ATTACK.HI             ANGLE OF ATTACK.LO
VERTICAL VELOCITY.HI           VERTICAL VELOCITY.LO
HEADING DEVIATION.HI           HEADING DEVIATION.LO
ROLL.HI                        ROLL.LO
PITCH.HI                       PITCH.LO
ALTITUDE DEVIATION.HI          ALTITUDE DEVIATION.LO
CHECKSUM.HI                    CHECKSUM.LO
ETX   (ASCII 3 - End of TeXt)
```

This format is actually a continuous stream of byte data from the ASCII character SOH to ETX. The data is transmitted at 9600 baud, 8 bits, no parity, and 1 stop bit. If the AOI calculates a different checksum of the packet from the received checksum, it will send STX for an another packet to be transmitted and will retry until a successful packet is received (Appendix C).

A data logging function is also performed by the FIP. The second serial port on the SBC-2 has been configured to stream data at approximately one frame per second. Under this system, no protocols are followed for the data logger. The frame is simply transmitted to the host device and it is assumed that the data arrived properly. Two tasks are utilized for this function. The foreground task services the AOI requests and the A/D system. The background task handles the once per second data logger update when the AOI does not have a pending request.

Downloading Instructions

The software running in the FIP is stored in the downloading computer system. To run the FIP code, a host computer must be used to send the program to the FIP computer. Any simple terminal program can be used to accomplish this task. During the development of the system, Mirror III, a common terminal emulation package was utilized. The FIP communicates at 9600 baud, 8 data bits, 1 stop bit, no parity. Communication is established through port A (JA) of the FIP computer system. When running, the system terminal will display <ok> on the screen. At this point, the system file called

FIPPRGM.TXT can be SEnt serially. Upon completion of the code transfer, the system is started by typing:

xxx GET.PSI.TO.ALT.OFFSET

where xxx is the current field altitude in feet. This entry runs a word that provides a curve fit that nulls the effect of temperature. Once an <ok> is received, type:

FIP_OUT FIP_OUT_TASK   (hit a carriage return)

This runs the background data streaming task. If a terminal is hooked to port B (JB) of the FIP computer, and the data is not streaming to the terminal, a download problem has occurred and a re-load is required after checking all the cable connections. When the terminal is streaming data, type:

FIP_AOI_TASK

and the AOI will be ready to access the FIP. For instructions on downloading the AOI code, see the AOI technical report refer...ced below.

ROM vs RAM

In this application, the system software is RAM based to allow maximum testing flexibility. ROMS were initially considered but the constant changes during flight made the RAM based system more appropriate. Initial tests on the system showed that the inclusion of RAM provided a readily changed system of code that could be modified in a standard PC editor environment without the need for a ROM burner. Once the FIP is operating in a static development state, the inclusion of ROMs is a simple task.

RESULTS

Initial flight tests of the system with the AOI provided real time data to both the AOI and the data monitoring terminal. For an explanation of the AOI inflight testing results, see the report titled Inflight Evaluation of an Acoustic Orientation Instrument Final Report, March 1990.

During the sensor testing phases of the project, cockpit readings were used to correlate sensor voltage readings to the actual flight values. In all cases the sensors provided accurate readings within the parameters specified in the Task Order. Because of the rapid changes that occurred during flight, it was difficult to quantify the accuracies of the system. It was found, however, that the FIP provided highly repetitive responses. Typical test protocols are shown below:

Altitude test sample

Initial flight level: 8,000 ft
Target flight level: 10,000 ft

The pilot is asked to maintain a flight level of 8,000 feet for a period of 30 seconds. At this point, the measurements are saved to a file on the data logger. The pilot is then asked to climb (typical climb rates were 200 ft/min) to an altitude of 10,000 ft. Data was acquired during the period with the experimenter noting the initial time, 1 minute intervals, and final time. During the same period, the pilot is asked to verbally report the altitude shown by the aircraft instruments in 100 foot intervals. Once altitude was stabilized, the file recording was ceased. Data files from the series were recorded and analyzed against the verbal reports. Similar series were taken for all altitudes.

Airspeed test sample

Minimum Airspeed: 90 kts
Maximum Airspeed: 180 kts

In this series, the pilot is asked to verbally report the airspeed shown on the aircraft instruments while slowly increasing speed over a period of several minutes. During the test, data are streamed to a portable computer and saved in a file. The operator notes the initial time, reported airspeed in 10 kt increments, and report time. This series was taken at multiple altitudes and in varying weather conditions.

Heading test sample

Heading was determined using the aircraft instrumentation as a reference. Static measurements taken on the ground were compared with the

readings from the cockpit instruments. While in flight, similar measurements were taken to evaluate the effect of flight dynamics and in-cabin interference.

Bank / Pitch Angles

Static measurement taken on the ground confirmed the operation of the gyro. While in flight, several maneuvers including rapid banking and pitch changes were accomplished to test the dynamics of the system. In all cases, the gyro performed to specifications.

All sensors, with the exception of the AOA vane, were tested during the individual flight tests of the FIP.

Appendix A -   **Airspeed to A/D Lookup Table**

Airspeed to PSI Table

Airspeed in Knots

airspeed kts	Pound/sq.foot	delta value	Absolute Airspeed
0.00	0.00		0.00
1.00	0.00	0.00	0.00
2.00	0.01	0.01	0.00
3.00	0.03	0.02	0.00
4.00	0.05	0.02	0.00
5.00	0.08	0.03	0.00
6.00	0.12	0.04	0.00
7.00	0.17	0.04	0.00
8.00	0.22	0.05	0.00
9.00	0.27	0.06	0.00
10.00	0.34	0.06	0.00
11.00	0.41	0.07	0.00
12.00	0.49	0.08	0.00
13.00	0.57	0.08	0.00
14.00	0.66	0.09	0.00
15.00	0.76	0.10	0.01
16.00	0.87	0.11	0.01
17.00	0.98	0.11	0.01
18.00	1.10	0.12	0.01
19.00	1.22	0.13	0.01
20.00	1.35	0.13	0.01
21.00	1.49	0.14	0.01
22.00	1.64	0.15	0.01
23.00	1.79	0.15	0.01
24.00	1.95	0.16	0.01
25.00	2.12	0.17	0.01
26.00	2.29	0.17	0.02
27.00	2.47	0.18	0.02
28.00	2.66	0.19	0.02
29.00	2.85	0.19	0.02
30.00	3.05	0.20	0.02
31.00	3.26	0.21	0.02
32.00	3.47	0.21	0.02
33.00	3.69	0.22	0.03
34.00	3.92	0.23	0.03
35.00	4.15	0.23	0.03
36.00	4.39	0.24	0.03
37.00	4.64	0.25	0.03
38.00	4.89	0.25	0.03
39.00	5.15	0.26	0.04
40.00	5.42	0.27	0.04
41.00	5.70	0.28	0.04
42.00	5.98	0.28	0.04
43.00	6.27	0.29	0.04
44.00	6.56	0.30	0.05

45.00	6.86	0.30	0.05
46.00	7.17	0.31	0.05
47.00	7.49	0.32	0.05
48.00	7.81	0.32	0.05
49.00	8.14	0.33	0.06
50.00	8.48	0.34	0.06
51.00	8.82	0.34	0.06
52.00	9.17	0.35	0.06
53.00	9.53	0.36	0.07
54.00	9.89	0.36	0.07
55.00	10.26	0.37	0.07
56.00	10.64	0.38	0.07
57.00	11.02	0.38	0.08
58.00	11.41	0.39	0.08
59.00	11.81	0.40	0.08
60.00	12.21	0.40	0.08
61.00	12.62	0.41	0.09
62.00	13.04	0.42	0.09
63.00	13.47	0.42	0.09
64.00	13.90	0.43	0.10
65.00	14.34	0.44	0.10
66.00	14.78	0.45	0.10
67.00	15.24	0.45	0.11
68.00	15.70	0.46	0.11
69.00	16.16	0.47	0.11
70.00	16.64	0.47	0.12
71.00	17.12	0.48	0.12
72.00	17.60	0.49	0.12
73.00	18.10	0.49	0.13
74.00	18.60	0.50	0.13
75.00	19.10	0.51	0.13
76.00	19.62	0.52	0.14
77.00	20.14	0.52	0.14
78.00	20.67	0.53	0.14
79.00	21.20	0.54	0.15
80.00	21.75	0.54	0.15
81.00	22.30	0.55	0.15
82.00	22.85	0.56	0.16
83.00	23.42	0.56	0.16
84.00	23.98	0.57	0.17
85.00	24.56	0.58	0.17
86.00	25.15	0.58	0.17
87.00	25.74	0.59	0.18
88.00	26.33	0.60	0.18
89.00	26.94	0.60	0.19
90.00	27.55	0.61	0.19
91.00	28.17	0.62	0.20
92.00	28.79	0.63	0.20
93.00	29.43	0.63	0.20
94.00	30.07	0.64	0.21
95.00	30.71	0.65	0.21
96.00	31.37	0.65	0.22

97.00	32.03	0.66	0.22
98.00	32.69	0.67	0.23
99.00	33.37	0.67	0.23
100.00	34.05	0.68	0.24
101.00	34.74	0.69	0.24
102.00	35.43	0.70	0.25
103.00	36.14	0.70	0.25
104.00	36.85	0.71	0.26
105.00	37.56	0.72	0.26
106.00	38.29	0.72	0.27
107.00	39.02	0.73	0.27
108.00	39.75	0.74	0.28
109.00	40.50	0.74	0.28
110.00	41.25	0.75	0.29
111.00	42.01	0.76	0.29
112.00	42.77	0.77	0.30
113.00	43.55	0.77	0.30
114.00	44.33	0.78	0.31
115.00	45.11	0.79	0.31
116.00	45.91	0.79	0.32
117.00	46.71	0.80	0.32
118.00	47.52	0.81	0.33
119.00	48.33	0.82	0.34
120.00	49.15	0.82	0.34
121.00	49.98	0.83	0.35
122.00	50.82	0.84	0.35
123.00	51.66	0.84	0.36
124.00	52.52	0.85	0.36
125.00	53.37	0.86	0.37
126.00	54.24	0.86	0.38
127.00	55.11	0.87	0.38
128.00	55.99	0.88	0.39
129.00	56.88	0.89	0.39
130.00	57.77	0.89	0.40
131.00	58.67	0.90	0.41
132.00	59.58	0.91	0.41
133.00	60.50	0.92	0.42
134.00	61.42	0.92	0.43
135.00	62.35	0.93	0.43
136.00	63.28	0.94	0.44
137.00	64.23	0.94	0.45
138.00	65.18	0.95	0.45
139.00	66.14	0.96	0.46
140.00	67.10	0.97	0.47
141.00	68.08	0.97	0.47
142.00	69.06	0.98	0.48
143.00	70.04	0.99	0.49
144.00	71.04	0.99	0.49
145.00	72.04	1.00	0.50
146.00	73.05	1.01	0.51
147.00	74.07	1.02	0.51
148.00	75.09	1.02	0.52

149.00	76.12	1.03	0.53
150.00	77.16	1.04	0.54
151.00	78.21	1.05	0.54
152.00	79.26	1.05	0.55
153.00	80.32	1.06	0.56
154.00	81.39	1.07	0.57
155.00	82.46	1.07	0.57
156.00	83.54	1.08	0.58
157.00	84.63	1.09	0.59
158.00	85.73	1.10	0.60
159.00	86.83	1.10	0.60
160.00	87.95	1.11	0.61
161.00	89.07	1.12	0.62
162.00	90.19	1.13	0.63
163.00	91.33	1.13	0.63
164.00	92.47	1.14	0.64
165.00	93.61	1.15	0.65
166.00	94.77	1.16	0.66
167.00	95.93	1.16	0.67
168.00	97.11	1.17	0.67
169.00	98.28	1.18	0.68
170.00	99.47	1.19	0.69
171.00	100.66	1.19	0.70
172.00	101.86	1.20	0.71
173.00	103.07	1.21	0.72
174.00	104.29	1.22	0.72
175.00	105.51	1.22	0.73
176.00	106.74	1.23	0.74
177.00	107.98	1.24	0.75
178.00	109.22	1.25	0.76
179.00	110.48	1.25	0.77
180.00	111.74	1.26	0.78
181.00	113.01	1.27	0.78
182.00	114.28	1.28	0.79
183.00	115.57	1.28	0.80
184.00	116.86	1.29	0.81
185.00	118.16	1.30	0.82
186.00	119.46	1.31	0.83
187.00	120.77	1.31	0.84
188.00	122.10	1.32	0.85
189.00	123.42	1.33	0.86
190.00	124.76	1.34	0.87
191.00	126.11	1.34	0.88
192.00	127.46	1.35	0.89
193.00	128.82	1.36	0.89
194.00	130.18	1.37	0.90
195.00	131.56	1.37	0.91
196.00	132.94	1.38	0.92
197.00	134.33	1.39	0.93
198.00	135.73	1.40	0.94
199.00	137.13	1.41	0.95
200.00	138.55	1.41	0.96

201.00	139.97	1.42	0.97
202.00	141.40	1.43	0.98
203.00	142.83	1.44	0.99
204.00	144.28	1.44	1.00
205.00	145.73	1.45	1.01
206.00	147.19	1.46	1.02
207.00	148.66	1.47	1.03
208.00	150.13	1.48	1.04
209.00	151.61	1.48	1.05
210.00	153.10	1.49	1.06
211.00	154.60	1.50	1.07
212.00	156.11	1.51	1.08
213.00	157.62	1.52	1.09
214.00	159.15	1.52	1.11
215.00	160.68	1.53	1.12
216.00	162.21	1.54	1.13
217.00	163.76	1.55	1.14
218.00	165.31	1.55	1.15
219.00	166.87	1.56	1.16
220.00	168.44	1.57	1.17
221.00	170.02	1.58	1.18
222.00	171.61	1.59	1.19
223.00	173.20	1.59	1.20
224.00	174.80	1.60	1.21
225.00	176.41	1.61	1.23
226.00	178.03	1.62	1.24
227.00	179.65	1.63	1.25
228.00	181.29	1.63	1.26
229.00	182.93	1.64	1.27
230.00	184.58	1.65	1.28
231.00	186.23	1.66	1.29
232.00	187.90	1.67	1.30
233.00	189.57	1.67	1.32
234.00	191.25	1.68	1.33
235.00	192.94	1.69	1.34
236.00	194.64	1.70	1.35
237.00	196.35	1.71	1.36
238.00	198.06	1.71	1.38
239.00	199.78	1.72	1.39
240.00	201.51	1.73	1.40
241.00	203.25	1.74	1.41
242.00	205.00	1.75	1.42
243.00	206.75	1.75	1.44
244.00	208.51	1.76	1.45
245.00	210.28	1.77	1.46
246.00	212.06	1.78	1.47
247.00	213.85	1.79	1.49
248.00	215.65	1.80	1.50
249.00	217.45	1.80	1.51
250.00	219.26	1.81	1.52
251.00	221.08	1.82	1.54
252.00	222.91	1.83	1.55

253.00	224.75	1.84	1.56
254.00	226.59	1.85	1.57
255.00	228.45	1.85	1.59
256.00	230.31	1.86	1.60
257.00	232.18	1.87	1.61
258.00	234.06	1.88	1.63
259.00	235.95	1.89	1.64
260.00	237.84	1.90	1.65
261.00	239.75	1.91	1.66
262.00	241.66	1.91	1.68
263.00	243.58	1.92	1.69
264.00	245.51	1.93	1.70
265.00	247.45	1.94	1.72
266.00	249.39	1.95	1.73
267.00	251.35	1.95	1.75
268.00	253.31	1.96	1.76
269.00	255.28	1.97	1.77
270.00	257.26	1.98	1.79
271.00	259.25	1.99	1.80
272.00	261.25	2.00	1.81
273.00	263.25	2.01	1.83
274.00	265.27	2.01	1.84
275.00	267.29	2.02	1.86
276.00	269.32	2.03	1.87
277.00	271.36	2.04	1.88
278.00	273.41	2.05	1.90
279.00	275.47	2.06	1.91
280.00	277.53	2.07	1.93
281.00	279.61	2.07	1.94
282.00	281.69	2.08	1.96
283.00	283.78	2.09	1.97
284.00	285.88	2.10	1.99
285.00	287.99	2.11	2.00
286.00	290.11	2.12	2.01
287.00	292.24	2.13	2.03
288.00	294.37	2.14	2.04
289.00	296.52	2.15	2.06
290.00	298.67	2.15	2.07
291.00	300.84	2.16	2.09
292.00	303.01	2.17	2.10
293.00	305.19	2.18	2.12
294.00	307.37	2.19	2.13
295.00	309.57	2.20	2.15
296.00	311.78	2.21	2.17
297.00	313.99	2.22	2.18
298.00	316.22	2.22	2.20
299.00	318.45	2.23	2.21
300.00	320.69	2.24	2.23
301.00	322.95	2.25	2.24
302.00	325.21	2.26	2.26
303.00	327.47	2.27	2.27
304.00	329.75	2.28	2.29

305.00	332.04	2.29	2.31
306.00	334.34	2.30	2.32
307.00	336.64	2.31	2.34
308.00	338.96	2.32	2.35
309.00	341.28	2.32	2.37
310.00	343.61	2.33	2.39
311.00	345.95	2.34	2.40
312.00	348.30	2.35	2.42
313.00	350.67	2.36	2.44
314.00	353.03	2.37	2.45
315.00	355.41	2.38	2.47
316.00	357.80	2.39	2.48
317.00	360.20	2.40	2.50
318.00	362.60	2.41	2.52
319.00	365.02	2.42	2.53
320.00	367.44	2.42	2.55
321.00	369.88	2.43	2.57
322.00	372.32	2.44	2.59
323.00	374.7	2.45	2.60
324.00	377.24	2.46	2.62
325.00	379.71	2.47	2.64
326.00	382.19	2.48	2.65
327.00	384.68	2.49	2.67
328.00	387.18	2.50	2.69
329.00	389.69	2.51	2.71
330.00	392.20	2.52	2.72
331.00	394.73	2.53	2.74
332.00	397.27	2.54	2.76
333.00	399.82	2.55	2.78
334.00	402.37	2.56	2.79
335.00	404.94	2.57	2.81
336.00	407.51	2.58	2.83
337.00	410.10	2.59	2.85
338.00	412.69	2.60	2.87
339.00	415.30	2.60	2.88
340.00	417.91	2.61	2.90
341.00	420.53	2.62	2.92
342.00	423.17	2.63	2.94
343.00	425.81	2.64	2.96
344.00	428.46	2.65	2.98
345.00	431.12	2.66	2.99
346.00	433.79	2.67	3.01
347.00	436.48	2.68	3.03
348.00	439.17	2.69	3.05
349.00	441.87	2.70	3.07
350.00	444.58	2.71	3.09
351.00	447.30	2.72	3.11
352.00	450.03	2.73	3.13
353.00	452.77	2.74	3.14
354.00	455.52	2.75	3.16
355.00	458.28	2.76	3.18
356.00	461.05	2.77	3.20

357.00	463.83	2.78	3.22
358.00	466.62	2.79	3.24
359.00	469.42	2.80	3.26
360.00	472.23	2.81	3.28
361.00	475.05	2.82	3.30
362.00	477.88	2.83	3.32
363.00	480.72	2.84	3.34
364.00	483.57	2.85	3.36
365.00	486.43	2.86	3.38
366.00	489.30	2.87	3.40
367.00	492.18	2.88	3.42
368.00	495.07	2.89	3.44
369.00	497.97	2.90	3.46
370.00	500.88	2.91	3.48
371.00	503.80	2.92	3.50
372.00	506.73	2.93	3.52
373.00	509.68	2.94	3.54
374.00	512.63	2.95	3.56
375.00	515.59	2.96	3.58
376.00	518.56	2.97	3.60
377.00	521.54	2.98	3.62
378.00	524.54	2.99	3.64
379.00	527.54	3.00	3.66
380.00	530.55	3.01	3.68
381.00	533.58	3.02	3.71
382.00	536.61	3.04	3.73
383.00	539.66	3.04	3.75
384.00	542.71	3.06	3.77
385.00	545.78	3.07	3.79
386.00	548.85	3.08	3.81
387.00	551.94	3.09	3.83
388.00	555.04	3.10	3.85
389.00	558.15	3.11	3.88
390.00	561.27	3.12	3.90
391.00	564.39	3.13	3.92
392.00	567.53	3.14	3.94
393.00	570.69	3.15	3.96
394.00	573.85	3.16	3.99
395.00	577.02	3.17	4.01
396.00	580.20	3.18	4.03
397.00	583.39	3.19	4.05
398.00	586.60	3.20	4.07
399.00	589.81	3.22	4.10

Appendix B -   **Altitude vs. PSI Code and Equation**

Altitude to PSI conversion code and equation

```
( PRESSURE TO ALTITUDE COVERSION TABLE GENERATOR AND )
( TABLE LOOKUP.  EQUATION IS DERIVED FROM BEST REGRESSION )
( FIT TO 1.000. )

( S. MIKITEN 12/28/88 )

ALSO MATH

DECIMAL

: WORD.ARRAY
  CREATE
    4 * ALLOT
  DOES>
    SWAP 4 * + ;

4620 WORD.ARRAY PSI.TO.ALT

: PSI.TO.ALT_GEN
  1052 590                    ( FROM 590 P,MB TO 1052 P,MB )
  DO                          ( GO FROM 14000 FEET TO -1000
FEET )
    10 0
    DO
      I I>F 10 I>F F/         ( GET FRACTIONAL VALUE I / 10 )
      J I>F F+                ( ADD J TO FRACTIONAL VALUE )
      FDUP                    ( DUPLICATE FOR USER IN
CONVERSION TO PSI )
      " 68.94752" FNUMBER?    ( PUT 68.94752 INTO F STACK )
      DROP                    ( DROP RESULT FROM FNUMBER? )
      F/                      ( DIVIDE J+I/10 BY 68.94752 TO
GET PSI )
      FDUP                    ( DUPLICATE TO USE IN REGRESSION
FIT )
      " 65631.615" FNUMBER?   ( PUT 65631.615 INTO F STACK )
      DROP                    ( DROP RESULT FROM FNUMBER? )
      FSWAP                   ( SWAP FOR 65631.615 PSI ON FP
STACK)
      " 12649.672" FNUMBER?   ( PUT 12649.672 INTO F STACK )
```

```
        DROP                      ( DROP RESULT FROM FNUMBER? )
        F*                        ( MULTIPLY PSI WITH 12649.672 )
        F-                        ( ALT = 65631.615 - 12649.672 *
PSI )
        FOVER                     ( GET A COPY OF PSI )
        FDUP                      ( GET ANOTHER COPY OF PSI )
        F*                        ( PSI^2 )
        " 1420.339" FNUMBER?      ( PUT 1420.339 INTO FP STACK )
        DROP                      ( DROP RESULT FROM FNUMBER? )
        F*
        F+                        ( ALT = ALT + 1420.339 * PSI^2 )
        FOVER                     ( GET A COPY OF PSI )
        FDUP                      ( GET ANOTHER COPY OF PSI )
        FDUP                      ( GET YET ANOTHER COPY OF PSI )
        F*
        F*                        ( RESULT IS PSI^3 )
        " 107.508" FNUMBER?       ( PUT 107.508 INTO FP STACK )
        DROP                      ( DROP RESULT FROM FNUMBER? )
        F*
        F-                        ( ALT = ALT - 107.508 * PSI^3 )
        FOVER
        FDUP
        FDUP
        FDUP
        F*
        F*
        F*                        ( RESULT IS PSI^4 )
        " 4.3969" FNUMBER?        ( PUT 4.3969 INTO FP STACO )
        DROP                      ( DROP RESULT FROM FNUMBER? )
        F*
        F+                        ( ALT = ALT - 4.3969 * PSI^4 )
        FOVER
        FDUP
        FD' P
        FDUP
        FDUP
        F*
        F*
        F*
        F*                        ( RESULT IS PSI^5 )
        " .073447" FNUMBER?       ( PUT .073447 INTO FP STACK )
        DROP
        F*
```

```
        F-                        ( ALT = ALT - .073447 * PSI^5 )
        F>I                       ( CONVERT FLOATING POINT VALUE TO
INTEGER )
        J 10 * I +                ( CALCULATE OFFSET INTO ARRAY )
        5900 -                    ( OFFSET FROM 5900 )
        PSI.TO.ALT                ( ADDRESS OF FPSI ARRAY )
        W!                        ( STORE ALT INTO FPSI TABLE )
        FDROP                     ( ELIMINATE PSI )
        FDROP                     ( ELIMINATE P,MB )
     LOOP
   LOOP ;


VARIABLE PSI.TO.ALT.OFFSET

: PSI.TO.ALT_LOOKUP
   2048 SWAP - I>F ( PUT PSI ONTO FLOATING POINT STACK )
   " 136.53333" FNUMBER? ( SCALE A/D VALUE TO PSI - 2048/15 )
   DROP                      ( DROP RESULT FROM FNUMBER? )
   F/                        ( DIVIDE A/D BY 273.06666 )
   " 68.94752" FNUMBER?  ( PUT PSI TO P,MB CONVERSTION INTO FP
STACK )
   DROP                      ( DROP RESULT FROM FNUMBER? )
   F*                        ( MULT BY 68.94752 FOR PRESSURE IN
MILLIBARS )
   " 10.0" FNUMBER?       ( PUT 10 INTO FP STACK )
   DROP                      ( DROP RESULT FROM FNUMBER? )
   F*                        ( MULTIPLY BY 10 TO OBTAIN OFFSET
VALUE )
   F>I                       ( CVT FROM FP TO INT )
   5900 -                    ( SUBTRACT 5900 FROM OFFSET INTO FPSI
TABLE )
   PSI.TO.ALT.OFFSET @ + ( ADJUST FOR BASE ALT. ERROR )
   DUP 0 <                   ( TEST IF RESULT IS LESS THAN ZERO )
   IF DROP 0 EXIT THEN   ( IF TRUE, MAKE RESULT = 0 )
   PSI.TO.ALT                ( THE FPSI ARRAY )
   W@                        ( FETCH THE VALUE FROM THE FPSI TABLE
)
   ;
```

Appendix C - **Source Code**

```
( FLIGHT PACK 11/13/89 )

( READS THE VME 750 TIME FROM THE NATIONAL MM58274 REAL TIME CHIP )
( WRITE A 1 AT F20001 TO START THE TIME )
( 1/1/89 SM )

HEX

: FIP ;

: GET.YEARS  ( - n   GETS YEARS FROM VME 750 )
   F2001B C@ ( FETCH TENS )
   0F AND    ( AND  VALUE )
   0A *      ( MULTIPLY IT BY 10 )
   F20019 C@ ( FETCH UNITS )
   0F AND
   + ;

: GET.MONTHS  ( - n   GETS MONTHS FROM VME 750 )
   F20017 C@ ( FETCH TENS )
   0F AND
   0A *      ( MULTIPLY IT BY 10 )
   F20015 C@ ( FETCH UNITS )
   0F AND
   + ;

: GET.DAY  ( - n  GETS THE DAY FROM THE VME 750 )
   F20013 C@ ( FETCH TENS )
   0F AND
   0A *      ( MULTIPLY IT BY 10 )
   F20011 C@ ( FETCH UNITS )
   0F AND
   + ;

: GET.HOURS ( - n   GETS THE HOURS FROM THE VME 750 )
   F2000F C@ ( FETCH TENS )
   0F AND
   0A *      ( MULTIPLY IT BY 10 )
   F2000D C@ ( FETCH UNITS )
   0F AND
   + ;

: GET.MINUTES ( - n  GETS THE MINUTES FROM THE VME 750 )
   F2000B C@ ( FETCH TENS )
   0F AND
   0A *      ( MULTIPLY IT BY 10 )
   F20009 C@ ( FETCH UNITS )
   0F AND
   + ;

: GET.SECONDS ( - n GETS THE SECONDS FROM THE VME 750 )
   F20007 C@ ( FETCH TENS )
   0F AND
   0A *      ( MULTIPLY IT BY 10 )
   F20005 C@ ( FETCH UNITS )
   0F AND
   + ;
```

```
: GET.TENTHS.OF.SECOND ( - n GETS THE 1/10TH SECONDS FROM THE VME 750 )
   F20003 C@ ( FETCH TENS )
   OF AND
   ;


: GET.TIME ( - n n n n n n  GETS TIME FROM THE VME750 BOARD )
   GET.TENTHS.OF.SECOND
   GET.SECONDS
   GET.MINUTES
   GET.HOURS
   GET.DAY
   GET.MONTHS
   GET.YEARS
   ;

VARIABLE SEPARATOR
3A SEPARATOR C! ( SET SEPARATOR TO A COLON )
VARIABLE BACKUP
8 BACKUP C!  ( BACKSPACE )

: SHOW.TIME ( n n n n n n n -  DISPLAYS TIME TO THE CURRENT DEVICE )
   GET.TIME
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . BACKUP C@ EMIT SEPARATOR C@ EMIT
   . CR ;

DECIMAL

: START GET.SECONDS . GET.TENTHS.OF.SECOND . CR ;
: STOP GET.SECONDS . GET.TENTHS.OF.SECOND . CR ;
: TESTIT 1000 0 DO 1 IF THEN LOOP ;
: BCM START TESTIT STOP ;

HEX

: VME750.SET.TIME
   1 F2001D C!   ( DAY OF WEEK - SUNDAY )
   8 F2001B C!   ( TENS OF YEAR - 89 )
   9 F20019 C!   ( DIGITS OF YEAR - 89 )
   0 F20017 C!   ( TENS OF MONTH - JUNE )
   6 F20015 C!   ( DIGITS OF MONTH - JUNE )
   0 F20013 C!   ( TENS OF DAY - 1 )
   1 F20011 C!   ( DIGITS OF DAY - 1 )
   2 F2000F C!   ( TENS OF HOUR - 21: 9 O'CLOCK )
   1 F2000D C!   ( DIGITS OF HOUR - 21: 9 O'CLOCK )
   1 F2000B C!   ( TENS OF MINUTES - 17 )
   7 F20009 C!   ( DIGITS OF MINUTES - 17 )
   0 F20007 C!   ( TENS OF SECONDS - 0 )
   0 F20005 C!   ( DIGITS OF SECONDS - 0 )
   0 F20003 C!   ( 10TH'S OF SECOND )
   1 F20001 C!   ( WRITE A 1 TO START REAL TIME CLOCK )
   ;

VME750.SET.TIME
```

```
DECIMAL

VARIABLE OUT_SECONDS
VARIABLE OUT_DELAY1
VARIABLE OUT_DELAY2
1000 OUT_DELAY1 !
410 OUT_DELAY2 !
0 OUT_SECONDS !

: OUT_DELAY
  OUT_DELAY1 @ 0
  DO
    OUT_DELAY2 @ 0
    DO
    LOOP
  LOOP
  OUT_SECONDS @
  1+
  OUT_SECONDS !
  ;


( PRESSURE TO ALTITUDE COVERSION TABLE GENERATOR AND )
( TABLE LOOKUP.  EQUATION IS DERIVED FROM BEST REGRESSION )
( FIT TO 1.000. )

( S. MIKITEN 12/28/88 )

ALSO MATH

DECIMAL

: WORD.ARRAY
  CREATE
    4 * ALLOT
  DOES>
    SWAP 4 * + ;

4620 WORD.ARRAY PSI.TO.ALT

: PSI.TO.ALT_GEN
  1052 590                 ( FROM 590 P,MB TO 1052 P,MB )
  DO                       ( GO FROM 14000 FEET TO -1000 FEET )
    10 0
    DO
      I I>F 10 I>F F/      ( GET FRACTIONAL VALUE I / 10 )
      J I>F F+            ( ADD J TO FRACTIONAL VALUE )
      FDUP                ( DUPLICATE FOR USER IN CONVERSION TO PSI )
      " 68.94752" FNUMBER? ( PUT 68.94752 INTO F STACK )
      DROP                ( DROP RESULT FROM FNUMBER? )
      F/                  ( DIVIDE J+I/10 BY 68.94752 TO GET PSI )
      FDUP                ( DUPLICATE TO USE IN REGRESSION FIT )
      " 65631.615" FNUMBER? ( PUT 65631.615 INTO F STACK )
      DROP                ( DROP RESULT FROM FNUMBER? )
      FSWAP               ( SWAP FOR 65631.615 PSI ON FP STACK)
      " 12649.672" FNUMBER? ( PUT 12649.672 INTO F STACK )
      DROP                ( DROP RESULT FROM FNUMBER? )
```

```
        F*                      ( MULTIPLY PSI WITH 12649.672 )
        F-                      ( ALT = 65631.615 - 12649.672 * PSI )
        FOVER                   ( GET A COPY OF PSI )
        FDUP                    ( GET ANOTHER COPY OF PSI )
        F*                      ( PSI^2 )
        " 1420.339" FNUMBER?    ( PUT 1420.339 INTO FP STACK )
        DROP                    ( DROP RESULT FROM FNUMBER? )
        F*
        F+                      ( ALT = ALT + 1420.339 * PSI^2 )
        FOVER                   ( GET A COPY OF PSI )
        FDUP                    ( GET ANOTHER COPY OF PSI )
        FDUP                    ( GET YET ANOTHER COPY OF PSI )
        F*
        F*                      ( RESULT IS PSI^3 )
        " 107.508" FNUMBER?     ( PUT 107.508 INTO FP STACK )
        DROP                    ( DROP RESULT FROM FNUMBER? )
        F*
        F-                      ( ALT = ALT - 107.508 * PSI^3 )
        FOVER
        FDUP
        FDUP
        FDUP
        F*
        F*
        F*                      ( RESULT IS PSI^4 )
        " 4.3969" FNUMBER?      ( PUT 4.3969 INTO FP STACO )
        DROP                    ( DROP RESULT FROM FNUMBER? )
        F*
        F+                      ( ALT = ALT - 4.3969 * PSI^4 )
        FOVER
        FDUP
        FDUP
        FDUP
        FDUP
        F*
        F*
        F*
        F*                      ( RESULT IS PSI^5 )
        " .073447" FNUMBER?     ( PUT .073447 INTO FP STACK )
        DROP
        F*
        F-                      ( ALT = ALT - .073447 * PSI^5 )
        F>I                     ( CONVERT FLOATING POINT VALUE TO INTEGER )
        J 10 * I +              ( CALCULATE OFFSET INTO ARRAY )
        5900 -                  ( OFFSET FROM 5900 )
        PSI.TO.ALT              ( ADDRESS OF FPSI ARRAY )
        W!                      ( STORE ALT INTO FPSI TABLE )
        FDROP                   ( ELIMINATE PSI )
        FDROP                   ( ELIMINATE P,MB )
    LOOP
  LOOP ;

VARIABLE PSI.TO.ALT.OFFSET

: PSI.TO.ALT_LOOKUP
  2048 SWAP - I>F ( PUT PSI ONTO FLOATING POINT STACK )
  " 136.53333" FNUMBER? ( SCALE A/D VALUE TO PSI - 2048/15 )
  DROP                      ( DROP RESULT FROM FNUMBER? )
```

```
F/                          ( DIVIDE A/D BY 273.06666 )
" 68.94752" FNUMBER?        ( PUT PSI TO P,MB CONVERSTION INTO FP STACK )
DROP                        ( DROP RESULT FROM FNUMBER? )
F*                          ( MULT BY 68.94752 FOR PRESSURE IN MILLIBARS )
" 10.0" FNUMBER?            ( PUT 10 INTO FP STACK )
DROP                        ( DROP RESULT FROM FNUMBER? )
F*                          ( MULTIPLY BY 10 TO OBTAIN OFFSET VALUE )
F>I                         ( CVT FROM FP TO INT )
5900 -                      ( SUBTRACT 5900 FROM OFFSET INTO FPSI TABLE )
PSI.TO.ALT.OFFSET @ +       ( ADJUST FOR BASE ALT. ERROR )
DUP 0 <                     ( TEST IF RESULT IS LESS THAN ZERO )
IF DROP 0 EXIT THEN         ( IF TRUE, MAKE RESULT = 0 )
PSI.TO.ALT                  ( THE FPSI ARRAY )
W@                          ( FETCH THE VALUE FROM THE FPSI TABLE )
;

FP

: LONG.ARRAY
  CREATE
    8 * ALLOT
  DOES>
    SWAP 8 * + ;

340 LONG.ARRAY PSI.TO.AIRSPEED

VARIABLE PSI.TO.AIRSPEED.PTR

0 PSI.TO.AIRSPEED.PTR !

: PSI.TO.AIRSPEED.GEN
  0.0069444 F*                      ( CONVERT TO PSI )
  PSI.TO.AIRSPEED.PTR @
  PSI.TO.AIRSPEED F!
  PSI.TO.AIRSPEED.PTR @
  1+ PSI.TO.AIRSPEED.PTR !
  ;

: PSI.TO.AIRSPEED.LOOKUP
  -1 PSI.TO.AIRSPEED.PTR !
  BEGIN
    PSI.TO.AIRSPEED.PTR @
    1+ PSI.TO.AIRSPEED.PTR !
    PSI.TO.AIRSPEED.PTR @
    331 >
    IF
      0 FDROP ( BCM MOD ) EXIT
    THEN
    FDUP
    PSI.TO.AIRSPEED.PTR @
    PSI.TO.AIRSPEED F@
    FSWAP ( BCM MOD )
    F>
  UNTIL
  FDROP
  PSI.TO.AIRSPEED.PTR @
  ;
```

```
( CALCULATIONS FOR MAGNETOMETER HEADING )

FVARIABLE L        ( 10.0 / 0.6 - 10 VOLTS/600 MILLIGAUSS )
FVARIABLE PT       ( PT=[PI/180]*P )
FVARIABLE RL       ( RL=[PI/180]*R )
FVARIABLE B1       ( B1=MAGNETOMETER X AXIS/L )
FVARIABLE B2       ( B2=MAGNETOMETER Y AXIS/L )
FVARIABLE B3       ( B3=MAGNETOMETER Z AXIS/L )
FVARIABLE B        ( B=SQRT[B1^2+B2^2+B3^2] )
FVARIABLE SD       ( SD=[B3*SIN[PT]+[B1*SIN[RL]-B2*COS[RL]]*COS[PT]]/B )
FVARIABLE CD       ( CD=SQRT[1-SD^2] )
FVARIABLE TD       ( TD=SD/CD )
FVARIABLE D        ( D=ATAN[TD] )
FVARIABLE CZ       ( CZ=[B3-B*SIN[PT]*SIN[D]]/[B*COS[PT]*COS[D]+.000001] )
FVARIABLE SZ       ( SZ=[B1*COS[RL]+B2*SIN[RL]]/[B*COS[D]] )
FVARIABLE TZ       ( TZ=SZ/[CZ+.000001] )
FVARIABLE Z        ( Z=ATAN[TZ] )
FVARIABLE MX
FVARIABLE MY
FVARIABLE MZ


: GET_HEADING { R P | }
  10.0 0.6 F/ L F!       ( 10.0 / .6 - 10 VOLTS/600 MILLIGAUSS )
  PI 180.0 F/ P I>F F* PT F!   ( PT=[PI/180]*P )
  PI 180.0 F/ R I>F F* RL F!   ( RL=[PI/180]*R )
  MX F@ L F@ F/ B1 F!          ( B1=MX/L )
  MY F@ L F@ F/ B2 F!          ( B2=MY/L )
  MZ F@ L F@ F/ B3 F!          ( B3=MZ/L )
  B1 F@ B1 F@ F*
  B2 F@ B2 F@ F*
  B3 F@ B3 F@ F*
  F+ F+ FSQRT B F!                    ( B=SQRT[B1^2+B2^2+B3^2] )
  B3 F@ PT F@ FSIN F*
  B1 F@ RL F@ FSIN F*
  B2 F@ RL F@ FCOS F*
  F- PT F@ FCOS F* F+ B F@ F/
  SD F!      ( SD=[B3*SIN[PT]+[B1*SIN[RL]-B2*COS[RL]]*COS[PT]]/B )
  1.0 SD F@ SD F@ F* F- FSQRT CD F!   ( CD=SQRT[1.-SD^2] )
  SD F@ CD F@ F/ TD F!                       ( TD=SD/CD )
  TD F@ FATAN D F!                           ( D=ATAN[TD] )
  B3 F@ B F@ PT F@ FSIN D F@ FSIN F* F* F-
  B F@ PT F@ FCOS D F@ FCOS F* F* 0.000001 F+ F/
  CZ F!      ( CZ=[B3-B*SIN[PT]*SIN[D]]/[B*COS[PT]*COS[D]+.000001] )
  B1 F@ RL F@ FCOS F*
  B2 F@ RL F@ FSIN F* F+
  B F@ D F@ FCOS F* F/
  SZ F!      ( SZ=[B1*COS[RL]+B2*SIN[RL]]/[B*COS[D]] )
  SZ F@ CZ F@ 0.000001 F+ F/
  TZ F!      ( TZ=SZ/[CZ+.000001] )
  TZ F@ FATAN Z F!   ( Z=ATAN[TZ] )
  CZ F@
  0.0 F<
  IF
    Z F@ PI F+ Z F!    ( IF CZ < 0 THEN Z = Z + PI )
  ELSE
    SZ F@
    0.0 F<
```

```
      IF
         Z F@ PI PI F* F+ Z F!    ( IF CZ > 0 AND SZ < 0 THEN Z = Z + PI*2 )
      THEN
    THEN
    180.0 PI F/ Z F@ F* ( AZIMUTH=[180/PI]*Z )
    F>I
    ;


INT

( LAST UPDATE 9/27/88 )
( BOARD NAME: MZ 8605 )
( AUTHOR: BCM )
( INPUT GAIN SET TO UNITY )
( SINGLE ENDED INPUT )
( + - 5V INPUT LEVELS )

HEX
FFFE00 CONSTANT A/D.BASE.ADDRESS

: START.CONVERSION  ( -  STARTS A CONVERSION - VALUE IGNORED )
   00 A/D.BASE.ADDRESS W! ;

: SELECT.CHANNEL   ( n - LOW 4 BITS USED OTHERS IGNORED, START CONVERT )
   A/D.BASE.ADDRESS 2 +   ( ADD TWO TO SET PROPER REGISTER )
   W!                ( WRITE THE CHANNEL NUMBER TO THE BOARD AND CONVERT )
   ;

: ANALOG.READ   ( - n   GET DIGITAL VALUE - LOWER 12 ARE VALID )
   A/D.BASE.ADDRESS 2 + W@ ;

: INT.LEVEL.SELECT   ( n - SELECT THE INTERRUPT LEVEL )
   A/D.BASE.ADDRESS 4 + W! DROP ( WRITE 8 BITS OF 16 BIT VALUE ) ;

: INT.ENABLE   ( n - ENABLE INTERRUPTS - VALUE IGNORED )
   A/D.BASE.ADDRESS 6 + W! ( SELECT THE PROPER ADDRESS AND WRITE )
   ;

: INT.DISABLE ( n - DISABLE ALL INTERRUPTS - VALUE IGNORED )
   A/D.BASE.ADDRESS 8 + W! ( SELECT PROPER ADDRESS AND WRITE )
   ;

: CONVERT.DONE? ( CH# - n  CHECK BIT IN ANALOG.READ FOR CIP CLEARED )
   BEGIN
    ANALOG.READ    ( GET VALUE ON STACK )
    8000 AND       ( AND WITH MOST SIGNIFICANT BIT )
    8000 = NOT     ( TEST IF HI BIT NOT CLEARED YET )
   UNTIL           ( CHECK UPPER BIT TILL NOT SET )
   ANALOG.READ     ( READ FINAL VALUE )
   0FFF AND        ( MASK OUT BITS 12-15 )
   ;

DECIMAL

0 CONSTANT CHANNEL.ZERO
1 CONSTANT CHANNEL.ONE
2 CONSTANT CHANNEL.TWO
3 CONSTANT CHANNEL.THREE
```

```forth
4 CONSTANT CHANNEL.FOUR
5 CONSTANT CHANNEL.FIVE
6 CONSTANT CHANNEL.SIX
7 CONSTANT CHANNEL.SEVEN
8 CONSTANT CHANNEL.EIGHT
9 CONSTANT CHANNEL.NINE
10 CONSTANT CHANNEL.TEN

: READ.IT ( n - n CHANNEL NUMBER SELECTED, VALUE OUT )
   SELECT.CHANNEL
   CONVERT.DONE? . CR ;

: TEST ( n - CHANNEL NUMBER )
  BEGIN
    DUP
    SELECT.CHANNEL
    CONVERT.DONE?
    . CR
    ?TERMINAL
  UNTIL
  DROP
 ;

( ARRAY DEFINITION WORDS )
( ARRAY SIZE LIMITED TO AVAILABLE VARIABLE SPACE IN MEMORY )
( LAST UPDATE: 12/28/88 )
( ARRAY VARIABLES HOLD THE BASE ADDRESS OF EACH ARRAY )

VARIABLE ARRAY.CH.0
VARIABLE ARRAY.CH.1
VARIABLE ARRAY.CH.2
VARIABLE ARRAY.CH.3
VARIABLE ARRAY.CH.4
VARIABLE ARRAY.CH.5
VARIABLE ARRAY.CH.6
VARIABLE ARRAY.CH.7
VARIABLE ARRAY.CH.8
VARIABLE ARRAY.CH.9
VARIABLE ARRAY.CH.10

VARIABLE FILTER_DEPTH
11 FILTER_DEPTH !

VARIABLE VTABLE
FILTER_DEPTH @ 2 * 4 - VALLOT

VARIABLE FIP_ARRAY_SIZE
6 FIP_ARRAY_SIZE !

VARIABLE OUT_ARRAY_SIZE
FIP_ARRAY_SIZE @ 11 + OUT_ARRAY_SIZE !

VARIABLE VFIP1
OUT_ARRAY_SIZE @ 1+ 2 * 4 - VALLOT

VARIABLE VFIP2
OUT_ARRAY_SIZE @ 1+ 2 * 4 - VALLOT
```

```
VARIABLE FLIP_FLOP
0 FLIP_FLOP !

VARIABLE AOI_CHECKSUM
0 AOI_CHECKSUM !

CHANNEL.ZERO CONSTANT ALTITUDE_SET_CHAN
CHANNEL.ONE CONSTANT HEADING_SET_CHAN
CHANNEL.FOUR CONSTANT ALTITUDE_CHAN
CHANNEL.FIVE CONSTANT AIR_SPEED_CHAN
CHANNEL.SIX CONSTANT PITCH_CHAN
CHANNEL.SEVEN CONSTANT ROLL_CHAN
CHANNEL.EIGHT CONSTANT Y_CHAN
CHANNEL.NINE CONSTANT X_CHAN
CHANNEL.TEN CONSTANT Z_CHAN

0 CONSTANT AIR_SPEED
1 CONSTANT ATTACK_ANGLE
2 CONSTANT VERTICAL_VELOCITY
3 CONSTANT HEADING_DEVIATION
4 CONSTANT ROLL
5 CONSTANT PITCH
6 CONSTANT ALTITUDE_DEVIATION

: VBUBBLE { TOP | TEMP -- }    ( BUBBLE SORT VTABLE )
   FILTER_DEPTH @ TOP 1+
   DO
      VTABLE I 2 * + W@
      VTABLE TOP 2 * + W@ <
      IF
         VTABLE TOP 2 * + W@ -> TEMP
         VTABLE I 2 * + W@ VTABLE TOP 2 * + W!
         TEMP VTABLE I 2 * + W!
      THEN
   LOOP ;

: VMEDIAN_FILTER ( -- N )           ( GET THE MEDIAN OF VTABLE )
   FILTER_DEPTH @ 1- 0
   DO
      I VBUBBLE
   LOOP
   VTABLE FILTER_DEPTH @ 2 / 2 * + W@ ;

: VTABLEFILL ( -- )     ( FILL VTABLE WITH KEYBOARD VALUES )
   FILTER_DEPTH @ 0
   DO
      KEY VTABLE I 2 * + W!
   LOOP ;

: VTABLEOUT              ( OUTPUT THE VTABLE )
   FILTER_DEPTH @ 0
   DO
      VTABLE I 2 * + W@ .
   LOOP
   CR ;

: WHICH.ARRAY ( LOC -- ADDR ) ( STORES AT CORRECT ARRAY ADDRESS )
   FLIP_FLOP @ 0 =
```

```
            IF
                VFIP1
            ELSE
                VFIP2
            THEN
            SWAP 2 * + W!
            ;


        : GET.WHICH.ARRAY ( LOC -- N )  ( FETCHES FROM CORRECT ARRAY )
            FLIP_FLOP @ 0 =
            IF
                VFIP1
            ELSE
                VFIP2
            THEN
            SWAP 2 * + W@
            ;


        : AOI.WHICH.ARRAY ( LOC -- N N )  ( FETCHES FROM CORRECT ARRAY )
            FLIP_FLOP @ 0 =
            IF
              VFIP1                 ( GET ADDRESS OF VFIP1 )
            ELSE
              VFIP2                 ( GET ADDRESS OF VFIP2 )
            THEN
            SWAP 2 * +              ( GENERATE OFFSET ADDRESS )
            DUP                     ( DUPLICATE ADDRESS )
            C@ SWAP 1+ C@           ( PUT HIGH AND LOW BYTE ONTO STACK )
            ;

        : GET_ALTITUDE_SET ( -- )
            FILTER_DEPTH @ 0 DO      ( GET FILTER_DEPTH SAMPLES )
              ALTITUDE_SET_CHAN SELECT.CHANNEL
              CONVERT.DONE?
              VTABLE I 2 * + W!
            LOOP
            VMEDIAN_FILTER
            2048 SWAP
            -
            I>F                     ( PUT HEADING ONTO FLOATING POINT STACK )
            " 6.202148438" FNUMBER?   ( PUT COUNTS PER VOLT INTO FP STACK )
            DROP                    ( DROP RESULT FROM FNUMBER? )
            F*                      ( MULTIPLY TO GET THOUSANDS OF FEET )
            F>I                     ( CONVERT RESULT TO INTEGER )
            ;

       FP

        : GET_RAW_AIR_SPEED ( -- AIR SPEED )
            FILTER_DEPTH @ 0 DO      ( GET FILTER_DEPTH SAMPLES )
              AIR_SPEED_CHAN SELECT.CHANNEL
              CONVERT.DONE?
              VTABLE I 2 * + W!
            LOOP
            VMEDIAN_FILTER
        ;

        : GET_AIR_SPEED ( -- )
```

```
  GET_RAW_AIR_SPEED
  642 -
  DUP 2048 > IF DROP 2048 THEN
  DUP 0 < IF DROP 0 THEN
  I>F 409.6 F/ 5.0 FSWAP F-
  PSI.TO.AIRSPEED.LOOKUP
  I>F 12.41 F* F>I
  AIR_SPEED WHICH.ARRAY
  ;

: GET_ATTACK_ANGLE
  0 ATTACK_ANGLE WHICH.ARRAY
  ;

VARIABLE VERTICAL_VELOCITY_SECONDS
VARIABLE VERTICAL_VELOCITY_FLAG
VARIABLE VERTICAL_VELOCITY_ALTITUDE
0 VERTICAL_VELOCITY_FLAG !
VARIABLE CURRENT_VERTICAL_VELOCITY
0 CURRENT_VERTICAL_VELOCITY !

: GET_VERTICAL_VELOCITY
  VERTICAL_VELOCITY_FLAG @ 0 =
  IF
     OUT_SECONDS @ 5 +
     VERTICAL_VELOCITY_SECONDS !
     1 VERTICAL_VELOCITY_FLAG !
     FILTER_DEPTH @ 0
     DO     ( GET FILTER_DEPTH SAMPLES )
       ALTITUDE_CHAN SELECT.CHANNEL
       CONVERT.DONE?
       VTABLE I 2 * + W!
     LOOP
     VMEDIAN_FILTER
     PSI.TO.ALT_LOOKUP
     VERTICAL_VELOCITY_ALTITUDE !
  ELSE
     OUT_SECONDS @
     VERTICAL_VELOCITY_SECONDS @
     =
     IF
        FILTER_DEPTH @ 0
        DO     ( GET FILTER_DEPTH SAMPLES )
          ALTITUDE_CHAN SELECT.CHANNEL
          CONVERT.DONE?
          VTABLE I 2 * + W!
        LOOP
        VMEDIAN_FILTER
        PSI.TO.ALT_LOOKUP
        VERTICAL_VELOCITY_ALTITUDE @
        -
        I>F
        12.0 F* F>I
        DUP
        2048 >
        IF
           DROP
           2048
```

```
        THEN
        DUP
        -2048 <
        IF
          DROP
          -2048
        THEN
        2048 +
        CURRENT_VERTICAL_VELOCITY !
        0 VERTICAL_VELOCITY_FLAG !
      THEN
    THEN
    CURRENT_VERTICAL_VELOCITY @
    VERTICAL_VELOCITY WHICH.ARRAY
    ;

: GET_HEADING_SET ( -- )
  FILTER_DEPTH @ 0
  DO      ( GET FILTER_DEPTH SAMPLES )
    HEADING_.LT_CHAN SELECT.CHANNEL
    CONVERT.DONE?
    VTABLE I 2 * + W!
  LOOP
  VMEDIAN_FILTER
  2048 SWAP
  -
  I>F                 ( PUT HEADING ONTO FLOATING POINT STACK )
  " 0.17578125" FNUMBER? ( PUT COUNTS PER VOLTS INTO FP STACK )
  DROP                ( DROP RESULT FROM FNUMBER? )
  F*                  ( MULTIPLY TO GET DEGREES )
  F>I                 ( CONVERT RESULT TO INTEGER )
  ;

: GET_X ( -- N )
    FILTER_DEPTH @ 0 DO     ( GET FILTER_DEPTH SAMPLES )
    X_CHAN SELECT.CHANNEL
    CONVERT.DONE?
    VTABLE I 2 * + W!
  LOOP
  VMEDIAN_FILTER
  DUP I>F
  2048 <
  IF
    " 204.8" FNUMBER? DROP F/
    " 10.0" FNUMBER? DROP F-
  ELSE
    " 2048.0" FNUMBER? DROP F-
    " 204.8" FNUMBER? DROP F/
  THEN
  MX F!
  ;

: GET_Y ( -- N )
    FILTER_DEPTH @ 0 DO     ( GET FILTER_DEPTH SAMPLES )
    Y_CHAN SELECT.CHANNEL
    CONVERT.DONE?
    VTABLE I 2 * + W!
  LOOP
```

```
        VMEDIAN_FILTER
        DUP I>F
        2048 <
        IF
          " 204.8" FNUMBER? DROP F/
          " 10.0" FNUMBER? DROP F-
        ELSE
          " 2048.0" FNUMBER? DROP F-
          " 204.8" FNUMBER? DROP F/
        THEN
        MY F!
        ;

    : GET_Z ( -- N )
        FILTER_DEPTH @ 0 DO     ( GET FILTER_DEPTH SAMPLES )
        Z_CHAN SELECT.CHANNEL
        CONVERT.DONE?
        VTABLE I 2 * + W!
      LOOP
      VMEDIAN_FILTER
      DUP I>F
      2048 <
      IF
        " 204.8" FNUMBER? DROP F/
        " 10.0" FNUMBER? DROP F-
      ELSE
        " 2048.0" FNUMBER? DROP F-
        " 204.8" FNUMBER? DROP F/
      THEN
      MZ F!
      ;

FP

VARIABLE STATIC_ROLL
VARIABLE MAX_LEFT_ROLL
VARIABLE MAX_RIGHT_ROLL
FVARIABLE LEFT_DEGREES
FVARIABLE RIGHT_DEGREES

2.7                     ( STATIC ROLL VOLTAGE )
204.8 F* 2048.0 F+ F>I ( SCALE ON 0-4096 OR -10 TO 10V )
STATIC_ROLL !
0.0                     ( MAX BANK LEFT VOLTAGE 0 V )
204.8 F* 2048.0 F+ F>I ( SCALE ON 0-4096 OR -10 TO 10V )
MAX_LEFT_ROLL !
7.4                     ( MAX RIGHT ROLL VOLTAGE )
204.8 F* 2048.0 F+ F>I ( SCALE ON 0-4096 OR -10V TO 10V )
MAX_RIGHT_ROLL !

STATIC_ROLL @
MAX_LEFT_ROLL @
- I>F 2048.0 FSWAP F/
LEFT_DEGREES F!         ( GET SCALING IN DEGREES )

MAX_RIGHT_ROLL @
STATIC_ROLL @
- I>F 2048.0 FSWAP F/
```

```
    RIGHT_DEGREES F!        ( GET SCALING IN DEGREES )

  : GET_ROLL ( -- )
    FILTER_DEPTH @ 0
    DO      ( GET FILTER_DEPTH SAMPLES )
      ROLL_CHAN SELECT.CHANNEL
      CONVERT.DONE?
      VTABLE I 2 * + W!
    LOOP
    VMEDIAN_FILTER
    DUP
    STATIC_ROLL @ -
    0 <
    IF
      MAX_LEFT_ROLL @ - I>F
      LEFT_DEGREES F@ F*
      F>I
    ELSE
      STATIC_ROLL @ - I>F
      RIGHT_DEGREES F@ F*
      F>I
      2048 +
    THEN
    DUP
    ROLL WHICH.ARRAY
    DUP
    2048 >
    IF
      2048 -
    THEN
    I>F
    " 29.25714286" FNUMBER? DROP
    F/
    F>I
    ROLL GET.WHICH.ARRAY
    2048 <
    IF
      DUP
      0 >
      IF
        70 SWAP -
      THEN
    ELSE
      -1 *
    THEN
    ;

VARIABLE STATIC_PITCH
VARIABLE MAX_DIVE_PITCH
VARIABLE MAX_CLIMB_PITCH
FVARIABLE DIVE_DEGREES
FVARIABLE CLIMB_DEGREES

2.73                    ( STATIC PITCH VOLTAGE )
204.8 F* 2048.0 F+ F>I ( SCALE ON 0-4096 OR -10 TO 10V)
STATIC_PITCH !
6.8                     ( MAX PITCH DIVE VOLTAGE )
204.8 F* 2048.0 F+ F>I ( SCALE ON 0-4096 OR -10 TO 10V )
```

```
MAX_DIVE_PITCH !
0.0
204.8 F* 2048.0 F+ F>I   ( SCALE ON 0-4096 OR -10 TO 10V )
MAX_CLIMB_PITCH !

STATIC_PITCH @
MAX_CLIMB_PITCH @
- I>F 2048.0 FSWAP F/
CLIMB_DEGREES F!          ( GET SCALING IN DEGREES )

MAX_DIVE_PITCH @
STATIC_PITCH @
- I>F 2048.0 FSWAP F/
DIVE_DEGREES F!          ( GET SCALING IN DEGREES )

: GET_PITCH ( -- )
  FILTER_DEPTH @ 0
  DO    ( GET FILTER_DEPTH SAMPLES )
    PITCH_CHAN SELECT.CHANNEL
    CONVERT.DONE?
    VTABLE I 2 * + W!
  LOOP
  VMEDIAN_FILTER
  DUP
  STATIC_PITCH @ -
  0 <
  IF
    MAX_CLIMB_PITCH @ - I>F
    CLIMB_DEGREES F@ F*
    F>I
  ELSE
    STATIC_PITCH @ - I>F
    DIVE_DEGREES F@ F*
    F>I
    2048 +
  THEN
  DUP
  PITCH WHICH.ARRAY
  DUP
  2048 >
  IF
    2048 -
  THEN
  I>F
  " 37.23636364" FNUMBER? DROP
  F/
  F>I
  PITCH GET.WHICH.ARRAY
  2048 <
  IF
    DUP
    0 >
    IF
      55  -
    THEN
  THEN
    .
```

```
INT

: GET_HEADING_DEVIATION ( R P -- N )
  GET_X
  GET_Y
  GET_Z
  GET_HEADING
  GET_HEADING_SET
  -
  360 MOD
  180 SWAP -
  HEADING_DEVIATION WHICH.ARRAY
   ;

: GET_ALTITUDE ( -- )
  FILTER_DEPTH @ 0
  DO    ( GET FILTER_DEPTH SAMPLES )
    ALTITUDE_CHAN SELECT.CHANNEL
    CONVERT.DONE?
    VTABLE I 2 * + W!
  LOOP
  VMEDIAN_FILTER
  ;

VARIABLE PSI.TO.ALT.PTR

FP

: GET.PSI.TO.ALT.OFFSET
  -1 PSI.TO.ALT.PTR !
  BEGIN
    DUP
    PSI.TO.ALT.PTR @
    1+ DUP
    PSI.TO.ALT.PTR !
    PSI.TO.ALT W@
    >
  UNTIL
  DROP
  PSI.TO.ALT.PTR @ 1-
  GET_ALTITUDE
  2048 SWAP - I>F
  136.53333 F/    ( 2048 / 15 PSI = 136.5333 )
  68.94752 F*     ( PSI TO P,MB )
  10.0 F*
  F>I
  5900 -
  -
  PSI.TO.ALT.OFFSET !
;

INT

: GET_ALTITUDE_DEVIATION ( -- )
  GET_ALTITUDE_SET
  GET_ALTITUDE
  -
  DUP
```

```
        2048 >
        IF
          DROP
          2048
        THEN
        DUP
        -2048 <
        IF
          DROP
          -2048
        THEN
        2048 +
        ALTITUDE_DEVIATION WHICH.ARRAY
        ;


: PUT_ARRAY ( LOC -- )
  FIP_ARRAY_SIZE @ 1+ + WHICH.ARRAY
  ;

: ACQUIRE_FIP ( -- )
  FLIP_FLOP @ 0 =
  IF
     1 FLIP_FLOP !
  ELSE
     0 FLIP_FLOP !
  THEN
  GET_ROLL
    DUP ROLL_CHAN PUT_ARRAY
  GET_PITCH
    DUP PITCH_CHAN PUT_ARRAY
  GET_HEADING_DEVIATION
  GET_ALTITUDE_DEVIATION
  GET_AIR_SPEED
  GET_ATTACK_ANGLE
  GET_VERTICAL_VELOCITY
  GET_ALTITUDE_SET
    ALTITUDE_SET_CHAN PUT_ARRAY
  GET_HEADING_SET
    HEADING_SET_CHAN PUT_ARRAY
  GET_ALTITUDE
    ALTITUDE_CHAN PUT_ARRAY
  GET_RAW_AIR_SPEED
    AIR_SPEED_CHAN PUT_ARRAY
  MX F@ F>I X_CHAN PUT_ARRAY
  MY F@ F>I Y_CHAN PUT_ARRAY
  MZ F@ F>I Z_CHAN PUT_ARRAY
  0 CHANNEL.TWO PUT_ARRAY
  0 CHANNEL.THREE PUT_ARRAY
  ;

( SET UP A SEPARATE USER AREA FOR AN EXTRA TERMINAL LINE )
( REFER TO MANUALS ON TERMINAL, BUILD, AND ACTIVATE )
( 1/1/89 SM )

1 CONSTANT CHANNEL.1
1000 2000 TERMINAL FIP_OUT ( THE TERMINAL )
CHANNEL.1 FIP_OUT BUILD    ( SET UP USER AREA FOR CHANNEL 1 )
```

```
VARIABLE AKEY

: FIP_AOI_TASK  ( -- )
  BEGIN
    AKEY #1 EXPECT       ( WAIT FOR A SINGLE KEY PRESS )
    AKEY @ 16777216 /    ( SHIFT RIGHT TO GET 8 BIT CHAR VALUE )
    2 =
    IF
      0 AOI_CHECKSUM !
      0 1 <# HOLD #> TYPE      ( OUTPUT SOH )
      0
      FIP_ARRAY_SIZE @ 1+ 0
      DO
        I AOI.WHICH.ARRAY
        2DUP                ( DUPLICATE HIGH AND LOW BYTES )
        AOI_CHECKSUM @     ( GET VALUE OF AOI_CHECKSUM )
        + + AOI_CHECKSUM ! ( ADD HIGH AND LOW BYTES TO AOI_CHECKSUM )
      LOOP
      AOI_CHECKSUM @ 65280 AND 256 /
      AOI_CHECKSUM @ 255 AND
      <# HOLD HOLD        ( OUTPUT AIRSPEED )
         HOLD HOLD        ( OUTPUT ANGLE OF ATTACK - NOT USED )
         HOLD HOLD        ( OUTPUT VERTICAL VELOCITY )
         HOLD HOLD        ( OUTPUT HEADING DEVIATION )
         HOLD HOLD        ( OUTPUT ROLL ANGLE )
         HOLD HOLD        ( OUTPUT PITCH ANGLE )
         HOLD HOLD        ( OUTPUT ALTITUDE DEVIATION )
         HOLD HOLD #> TYPE ( OUTPUT CHECKSUM )
      0 3 <# HOLD #> TYPE      ( OUTPUT ETX )
    THEN
  AGAIN
  ;    ( LOOP FOREVER )


: FIP_OUT_TASK  ( -- )
  ACTIVATE
  ACQUIRE_FIP
  BEGIN
    OUT_DELAY
    OUT_ARRAY_SIZE @ 1+ 0
    DO
       I GET.WHICH.ARRAY .
    LOOP
    CR
    ACQUIRE_FIP
  AGAIN
  ;    ( LOOP UNTIL KEYPRESS )

FP

0.0 PSI.TO.AIRSPEED.GEN
.003598 PSI.TO.AIRSPEED.GEN
.013332 PSI.TO.AIRSPEED.GEN
.030262 PSI.TO.AIRSPEED.GEN
.053964 PSI.TO.AIRSPEED.GEN
.084437 PSI.TO.AIRSPEED.GEN
.122106 PSI.TO.AIRSPEED.GEN
```

```
.165911 PSI.TO.AIRSPEED.GEN
.216912 PSI.TO.AIRSPEED.GEN
.274050 PSI.TO.AIRSPEED.GEN
.338383 PSI.TO.AIRSPEED.GEN
.409488 PSI.TO.AIRSPEED.GEN
.487365 PSI.TO.AIRSPEED.GEN
.571802 PSI.TO.AIRSPEED.GEN
.663646 PSI.TO.AIRSPEED.GEN
.761415 PSI.TO.AIRSPEED.GEN
.866591 PSI.TO.AIRSPEED.GEN
.978327 PSI.TO.AIRSPEED.GEN
1.09684 PSI.TO.AIRSPEED.GEN
1.22233 PSI.TO.AIRSPEED.GEN
1.35438 PSI.TO.AIRSPEED.GEN
1.49363 PSI.TO.AIRSPEED.GEN
1.63880 PSI.TO.AIRSPEED.GEN
1.79159 PSI.TO.AIRSPEED.GEN
1.95073 PSI.TO.AIRSPEED.GEN
2.11685 PSI.TO.AIRSPEED.GEN
2.28954 PSI.TO.AIRSPEED.GEN
2.46899 PSI.TO.AIRSPEED.GEN
2.65585 PSI.TO.AIRSPEED.GEN
2.84843 PSI.TO.AIRSPEED.GEN
3.04883 PSI.TO.AIRSPEED.GEN
3.25559 PSI.TO.AIRSPEED.GEN
3.46890 PSI.TO.AIRSPEED.GEN
3.68941 PSI.TO.AIRSPEED.GEN
3.91648 PSI.TO.AIRSPEED.GEN
4.14990 PSI.TO.AIRSPEED.GEN
4.39115 PSI.TO.AIRSPEED.GEN
4.63896 PSI.TO.AIRSPEED.GEN
4.89248 PSI.TO.AIRSPEED.GEN
5.15362 PSI.TO.AIRSPEED.GEN
5.42154 PSI.TO.AIRSPEED.GEN
5.69686 PSI.TO.AIRSPEED.GEN
5.97831 PSI.TO.AIRSPEED.GEN
6.26675 PSI.TO.AIRSPEED.GEN
6.56175 PSI.TO.AIRSPEED.GEN
6.86374 PSI.TO.AIRSPEED.GEN
7.17249 PSI.TO.AIRSPEED.GEN
7.48781 PSI.TO.AIRSPEED.GEN
7.81032 PSI.TO.AIRSPEED.GEN
8.14003 PSI.TO.AIRSPEED.GEN
8.47587 PSI.TO.AIRSPEED.GEN
8.81891 PSI.TO.AIRSPEED.GEN
9.16893 PSI.TO.AIRSPEED.GEN
9.52552 PSI.TO.AIRSPEED.GEN
9.88908 PSI.TO.AIRSPEED.GEN
10.2590 PSI.TO.AIRSPEED.GEN
10.6359 PSI.TO.AIRSPEED.GEN
11.0202 PSI.TO.AIRSPEED.GEN
11.4115 PSI.TO.AIRSPEED.GEN
11.8083 PSI.TO.AIRSPEED.GEN
12.2129 PSI.TO.AIRSPEED.GEN
12.6241 PSI.TO.AIRSPEED.GEN
13.0431 PSI.TO.AIRSPEED.GEN
13.4678 PSI.TO.AIRSPEED.GEN
13.8995 PSI.TO.AIRSPEED.GEN
```

```
14.3384 PSI.TO.AIRSPEED.GEN
14.7845 PSI.TO.AIRSPEED.GEN
15.2368 PSI.TO.AIRSPEED.GEN
15.6962 PSI.TO.AIRSPEED.GEN
16.1624 PSI.TO.AIRSPEED.GEN
16.6352 PSI.TO.AIRSPEED.GEN
17.1158 PSI.TO.AIRSPEED.GEN
17.6029 PSI.TO.AIRSPEED.GEN
18.0964 PSI.TO.AIRSPEED.GEN
18.5971 PSI.TO.AIRSPEED.GEN
19.1046 PSI.TO.AIRSPEED.GEN
19.6201 PSI.TO.AIRSPEED.GEN
20.1409 PSI.TO.AIRSPEED.GEN
20.6691 PSI.TO.AIRSPEED.GEN
21.2045 PSI.TO.AIRSPEED.GEN
21.7471 PSI.TO.AIRSPEED.GEN
22.2963 PSI.TO.AIRSPEED.GEN
22.8522 PSI.TO.AIRSPEED.GEN
23.4151 PSI.TO.AIRSPEED.GEN
23.9846 PSI.TO.AIRSPEED.GEN
24.5619 PSI.TO.AIRSPEED.GEN
25.1459 PSI.TO.AIRSPEED.GEN
25.7364 PSI.TO.AIRSPEED.GEN
26.3338 PSI.TO.AIRSPEED.GEN
26.9386 PSI.TO.AIRSPEED.GEN
27.5500 PSI.TO.AIRSPEED.GEN
28.1690 PSI.TO.AIRSPEED.GEN
28.7941 PSI.TO.AIRSPEED.GEN
29.4268 PSI.TO.AIRSPEED.GEN
30.0664 PSI.TO.AIRSPEED.GEN
30.7129 PSI.TO.AIRSPEED.GEN
31.3664 PSI.TO.AIRSPEED.GEN
32.0266 PSI.TO.AIRSPEED.GEN
32.6936 PSI.TO.AIRSPEED.GEN
33.3685 PSI.TO.AIRSPEED.GEN
34.0493 PSI.TO.AIRSPEED.GEN
34.7379 PSI.TO.AIRSPEED.GEN
35.4333 PSI.TO.AIRSPEED.GEN
36.1357 PSI.TO.AIRSPEED.GEN
36.8450 PSI.TO.AIRSPEED.GEN
37.5612 PSI.TO.AIRSPEED.GEN
38.2853 PSI.TO.AIRSPEED.GEN
39.0152 PSI.TO.AIRSPEED.GEN
39.7529 PSI.TO.AIRSPEED.GEN
40.4976 PSI.TO.AIRSPEED.GEN
41.2493 PSI.TO.AIRSPEED.GEN
42.0078 PSI.TO.AIRSPEED.GEN
42.7740 PSI.TO.AIRSPEED.GEN
43.5467 PSI.TO.AIRSPEED.GEN
44.3269 PSI.TO.AIRSPEED.GEN
45.1131 PSI.TO.AIRSPEED.GEN
45.9073 PSI.TO.AIRSPEED.GEN
46.7085 PSI.TO.AIRSPEED.GEN
47.5167 PSI.TO.AIRSPEED.GEN
48.3325 PSI.TO.AIRSPEED.GEN
49.1544 PSI.TO.AIRSPEED.GEN
49.9844 PSI.TO.AIRSPEED.GEN
50.8212 PSI.TO.AIRSPEED.GEN
```

```
51.6643 PSI.TO.AIRSPEED.GEN
52.5150 PSI.TO.AIRSPEED.GEN
53.3737 PSI.TO.AIRSPEED.GEN
54.2386 PSI.TO.AIRSPEED.GEN
55.1107 PSI.TO.AIRSPEED.GEN
55.9904 PSI.TO.AIRSPEED.GEN
56.8774 PSI.TO.AIRSPEED.GEN
57.7710 PSI.TO.AIRSPEED.GEN
58.6717 PSI.TO.AIRSPEED.GEN
59.5800 PSI.TO.AIRSPEED.GEN
60.4952 PSI.TO.AIRSPEED.GEN
61.4175 PSI.TO.AIRSPEED.GEN
62.3471 PSI.TO.AIRSPEED.GEN
63.2844 PSI.TO.AIRSPEED.GEN
64.2282 PSI.TO.AIRSPEED.GEN
65.1797 PSI.TO.AIRSPEED.GEN
66.1381 PSI.TO.AIRSPEED.GEN
67.1035 PSI.TO.AIRSPEED.GEN
68.0762 PSI.TO.AIRSPEED.GEN
69.0566 PSI.TO.AIRSPEED.GEN
70.0447 PSI.TO.AIRSPEED.GEN
71.0391 PSI.TO.AIRSPEED.GEN
72.0411 PSI.TO.AIRSPEED.GEN
73.0501 PSI.TO.AIRSPEED.GEN
74.0665 PSI.TO.AIRSPEED.GEN
74.0899 PSI.TO.AIRSPEED.GEN
76.1212 PSI.TO.AIRSPEED.GEN
77.1598 PSI.TO.AIRSPEED.GEN
78.2056 PSI.TO.AIRSPEED.GEN
79.2576 PSI.TO.AIRSPEED.GEN
80.3187 PSI.TO.AIRSPEED.GEN
81.3861 PSI.TO.AIRSPEED.GEN
82.4607 PSI.TO.AIRSPEED.GEN
83.5434 PSI.TO.AIRSPEED.GEN
84.6630 PSI.TO.AIRSPEED.GEN
85.7294 PSI.TO.AIRSPEED.GEN
86.8337 PSI.TO.AIRSPEED.GEN
87.9462 PSI.TO.AIRSPEED.GEN
89.0650 PSI.TO.AIRSPEED.GEN
90.1911 PSI.TO.AIRSPEED.GEN
91.3253 PSI.TO.AIRSPEED.GEN
92.4658 PSI.TO.AIRSPEED.GEN
93.6147 PSI.TO.AIRSPEED.GEN
94.7705 PSI.TO.AIRSPEED.GEN
95.9340 PSI.TO.AIRSPEED.GEN
97.1054 PSI.TO.AIRSPEED.GEN
98.2835 PSI.TO.AIRSPEED.GEN
99.4696 PSI.TO.AIRSPEED.GEN
100.662 PSI.TO.AIRSPEED.GEN
101.863 PSI.TO.AIRSPEED.GEN
103.071 PSI.TO.AIRSPEED.GEN
104.287 PSI.TO.AIRSPEED.GEN
105.510 PSI.TO.AIRSPEED.GEN
106.741 PSI.TO.AIRSPEED.GEN
107.978 PSI.TO.AIRSPEED.GEN
109.224 PSI.TO.AIRSPEED.GEN
110.478 PSI.TO.AIRSPEED.GEN
111.738 PSI.TO.AIRSPEED.GEN
```

```
113.006 PSI.TO.AIRSPEED.GEN
114.282 PSI.TO.AIRSPEED.GEN
115.565 PSI.TO.AIRSPEED.GEN
116.856 PSI.TO.AIRSPEED.GEN
118.155 PSI.TO.AIRSPEED.GEN
119.460 PSI.TO.AIRSPEED.GEN
120.774 PSI.TO.AIRSPEED.GEN
122.096 PSI.TO.AIRSPEED.GEN
123.424 PSI.TO.AIRSPEED.GEN
124.761 PSI.TO.AIRSPEED.GEN
126.105 PSI.TO.AIRSPEED.GEN
127.456 PSI.TO.AIRSPEED.GEN
128.816 PSI.TO.AIRSPEED.GEN
130.183 PSI.TO.AIRSPEED.GEN
130.557 PSI.TO.AIRSPEED.GEN
132.939 PSI.TO.AIRSPEED.GEN
134.330 PSI.TO.AIRSPEED.GEN
135.727 PSI.TO.AIRSPEED.GEN
137.133 PSI.TO.AIRSPEED.GEN
138.546 PSI.TO.AIRSPEED.GEN
139.967 PSI.TO.AIRSPEED.GEN
141.396 PSI.TO.AIRSPEED.GEN
142.832 PSI.TO.AIRSPEED.GEN
144.276 PSI.TO.AIRSPEED.GEN
145.728 PSI.TO.AIRSPEED.GEN
147.187 PSI.TO.AIRSPEED.GEN
148.655 PSI.TO.AIRSPEED.GEN
150.130 PSI.TO.AIRSPEED.GEN
151.613 PSI.TO.AIRSPEED.GEN
153.103 PSI.TO.AIRSPEED.GEN
154.602 PSI.TO.AIRSPEED.GEN
156.108 PSI.TO.AIRSPEED.GEN
157.623 PSI.TO.AIRSPEED.GEN
159.145 PSI.TO.AIRSPEED.GEN
160.675 PSI.TO.AIRSPEED.GEN
162.213 PSI.TO.AIRSPEED.GEN
163.759 PSI.TO.AIRSPEED.GEN
165.312 PSI.TO.AIRSPEED.GEN
166.874 PSI.TO.AIRSPEED.GEN
168.443 PSI.TO.AIRSPEED.GEN
170.020 PSI.TO.AIRSPEED.GEN
171.606 PSI.TO.AIRSPEED.GEN
173.199 PSI.TO.AIRSPEED.GEN
174.800 PSI.TO.AIRSPEED.GEN
176.410 PSI.TO.AIRSPEED.GEN
178.027 PSI.TO.AIRSPEED.GEN
179.652 PSI.TO.AIRSPEED.GEN
181.285 PSI.TO.AIRSPEED.GEN
182.927 PSI.TO.AIRSPEED.GEN
184.576 PSI.TO.AIRSPEED.GEN
186.233 PSI.TO.AIRSPEED.GEN
187.898 PSI.TO.AIRSPEED.GEN
189.572 PSI.TO.AIRSPEED.GEN
191.252 PSI.TO.AIRSPEED.GEN
192.942 PSI.TO.AIRSPEED.GEN
194.640 PSI.TO.AIRSPEED.GEN
196.346 PSI.TO.AIRSPEED.GEN
198.059 PSI.TO.AIRSPEED.GEN
```

```
199.781 PSI.TO.AIRSPEED.GEN
201.511 PSI.TO.AIRSPEED.GEN
203.250 PSI.TO.AIRSPEED.GEN
204.996 PSI.TO.AIRSPEED.GEN
206.750 PSI.TO.AIRSPEED.GEN
208.513 PSI.TO.AIRSPEED.GEN
210.284 PSI.TO.AIRSPEED.GEN
212.064 PSI.TO.AIRSPEED.GEN
213.851 PSI.TO.AIRSPEED.GEN
215.647 PSI.TO.AIRSPEED.GEN
217.451 PSI.TO.AIRSPEED.GEN
219.262 PSI.TO.AIRSPEED.GEN
221.083 PSI.TO.AIRSPEED.GEN
222.912 PSI.TO.AIRSPEED.GEN
224.749 PSI.TO.AIRSPEED.GEN
226.594 PSI.TO.AIRSPEED.GEN
228.448 PSI.TO.AIRSPEED.GEN
230.310 PSI.TO.AIRSPEED.GEN
232.181 PSI.TO.AIRSPEED.GEN
234.059 PSI.TO.AIRSPEED.GEN
235.946 PSI.TO.AIRSPEED.GEN
237.841 PSI.TO.AIRSPEED.GEN
239.746 PSI.TO.AIRSPEED.GEN
241.657 PSI.TO.AIRSPEED.GEN
243.578 PSI.TO.AIRSPEED.GEN
245.507 PSI.TO.AIRSPEED.GEN
247.445 PSI.TO.AIRSPEED.GEN
249.391 PSI.TO.AIRSPEED.GEN
251.345 PSI.TO.AIRSPEED.GEN
252.305 PSI.TO.AIRSPEED.GEN
255.280 PSI.TO.AIRSPEED.GEN
257.260 PSI.TO.AIRSPEED.GEN
259.249 PSI.TO.AIRSPEED.GEN
261.246 PSI.TO.AIRSPEED.GEN
263.252 PSI.TO.AIRSPEED.GEN
265.266 PSI.TO.AIRSPEED.GEN
267.289 PSI.TO.AIRSPEED.GEN
269.320 PSI.TO.AIRSPEED.GEN
271.361 PSI.TO.AIRSPEED.GEN
273.409 PSI.TO.AIRSPEED.GEN
275.467 PSI.TO.AIRSPEED.GEN
277.533 PSI.TO.AIRSPEED.GEN
279.607 PSI.TO.AIRSPEED.GEN
281.691 PSI.TO.AIRSPEED.GEN
283.782 PSI.TO.AIRSPEED.GEN
285.884 PSI.TO.AIRSPEED.GEN
287.993 PSI.TO.AIRSPEED.GEN
290.111 PSI.TO.AIRSPEED.GEN
292.238 PSI.TO.AIRSPEED.GEN
295.374 PSI.TO.AIRSPEED.GEN
296.519 PSI.TO.AIRSPEED.GEN
298.672 PSI.TO.AIRSPEED.GEN
300.835 PSI.TO.AIRSPEED.GEN
303.006 PSI.TO.AIRSPEED.GEN
305.186 PSI.TO.AIRSPEED.GEN
307.374 PSI.TO.AIRSPEED.GEN
309.572 PSI.TO.AIRSPEED.GEN
311.779 PSI.TO.AIRSPEED.GEN
```

```
313.994 PSI.TO.AIRSPEED.GEN
316.218 PSI.TO.AIRSPEED.GEN
318.542 PSI.TO.AIRSPEED.GEN
320.694 PSI.TO.AIRSPEED.GEN
322.945 PSI.TO.AIRSPEED.GEN
325.205 PSI.TO.AIRSPEED.GEN
327.474 PSI.TO.AIRSPEED.GEN
329.753 PSI.TO.AIRSPEED.GEN
332.040 PSI.TO.AIRSPEED.GEN
334.336 PSI.TO.AIRSPEED.GEN
336.641 PSI.TO.AIRSPEED.GEN
338.956 PSI.TO.AIRSPEED.GEN
341.280 PSI.TO.AIRSPEED.GEN
343.612 PSI.TO.AIRSPEED.GEN
345.954 PSI.TO.AIRSPEED.GEN
348.302 PSI.TO.AIRSPEED.GEN
350.665 PSI.TO.AIRSPEED.GEN
353.034 PSI.TO.AIRSPEED.GEN
355.413 PSI.TO.AIRSPEED.GEN
357.801 PSI.TO.AIRSPEED.GEN
360.197 PSI.TO.AIRSPEED.GEN
362.604 PSI.TO.AIRSPEED.GEN
365.019 PSI.TO.AIRSPEED.GEN
367.443 PSI.TO.AIRSPEED.GEN
369.887 PSI.TO.AIRSPEED.GEN
372.320 PSI.TO.AIRSPEED.GEN
374.773 PSI.TO.AIRSPEED.GEN
377.235 PSI.TO.AIRSPEED.GEN
379.706 PSI.TO.AIRSPEED.GEN
382.187 PSI.TO.AIRSPEED.GEN
384.677 PSI.TO.AIRSPEED.GEN
387.177 PSI.TO.AIRSPEED.GEN
389.685 PSI.TO.AIRSPEED.GEN
392.203 PSI.TO.AIRSPEED.GEN

INT


PSI.TO.ALT_GEN
( X GET.PSI.TO.ALT.OFFSET  --- X IS CURRENT ALTITUDE )
( FIP_OUT FIP_OUT_TASK )
( FIP_AOI_TASK )
```

Appendix D -   **Vertical Gyroscope**

Appendix D -   Humphrey Inc. Data

ELECTRO-MECHANICAL INSTRUMENTS

Humphrey
SAN DIEGO CALIFORNIA

TITLE: ENVELOPE DRAWING DIRECTIONAL GYRO & FLUX GATE SYSTEM

MODEL NO: DG11-0105-1

C

REV: F

SPECIFICATIONS

1.0	MECHANICAL FREEDOM	
*1.1	INNER GIMBAL	±85° MINIMUM
*1.2	OUTER GIMBAL	360° CONTINUOUS
*2.0	ACCURACY (WITH FLUX DETECTOR)	±1.5°
3.0	SPIN MOTOR AND ERECTION SYSTEM	
*3.1	INPUT MOTOR	115V ±10%, 65 MA, 400 ±5% HZ NOMINAL
3.2	ERECTION SYSTEM	26V ±10%, 50 MA, 400 ±5% HZ NOMINAL
4.0	AMPLIFIER (SUPPLIED FROM 26V INPUT)	
4.1	VOLTAGE	28 VDC
4.2	POWER	4 WATTS APPROXIMATELY
5.0	OUTPUT SYNCHRO OUTER GIMBAL	
5.1	INPUT VOLTAGE	
*5.2	INPUT VOLTAGE, 400 HZ	26 VOLTS
*5.3	OUTPUT VOLTAGE	11.8 VOLTS
5.4	ACCURACY	±0.5°
*5.5	NULL POSITION	NULL IS SET WHEN FORE POINTER ON FLUX GATE AND FORE POSITION ON GYRO ARE PARALLELED AND GYRO IS SLAVED.
6.0	SLAVING RATE	0.5°/MIN. MINIMUM
7.0	INSULATION RESISTANCE	50 MEGOHMS MIN BETWEEN ANY PIN AND INSTRUMENT WITH 50 VDC APPLIED
8.0	SERVICE LIFE	500 HOURS MINIMUM
9.0	WEIGHT	3.3 POUNDS
10.0	ENVIRONMENTAL CONDITIONS	
10.1	ALTITUDE	-1000 FEET TO 40,000 FEET
10.2	TEMPERATURE	-55°C TO +70°C
10.3	VIBRATION	0.036 DA OR 5G, WHICHEVER IS LIMITING, 5 TO 500 HZ
10.4	SHOCK	15G, 11 MS
10.5	HUMIDITY	0 TO 95% RELATIVE HUMIDITY AT 32°C
11.0	REMARKS	
11.1	ITEMS MARKED WITH (*) ARE CHECKED IN PRODUCTION TESTS. OTHER ITEMS FOR REFERENCE MAY BE CHECKED ON ORDER BY QUALIFICATION TESTS.	
11.2	FOR APPLICATION, SEE DWG. A-31098 OR BLOCK DIAGRAM B-32338	
11.3	INSTRUMENT MANUFACTURED IN ACCORDANCE WITH TSO C6$_c$ DATED 1 APRIL 1959	

SPECIFICATIONS

1.0	RANGE	±40°/SECOND
•2.0	NATURAL FREQUENCY	10 Hz APPROXIMATELY
3.0	POTENTIOMETER	
•3.1	RESISTANCE	1,000 OHMS ±10%
3.2	POWER DISSIPATION	0.5 WATTS MAXIMUM
•4.0	ACCURACY (RATE RANGE)	±1.0% OF FULL SCALE
		MAXIMUM RATE.
•5.0	REPEATABILITY AND HYSTERESIS	1.0% OF FULL SCALE MAXIMUM
•6.0	THRESHOLD	1.0% OF FULL SCALE MAXIMUM
•7.0	DAMPING FACTOR	0.5 ±0.2 OF CRITICAL OVER TEMPERATURE
		EXTREMES
•8.0	AXIS ALIGNMENT	0.5°
9.0	MOTOR	
•9.1	VOLTAGE	28 VDC, ±10%
•9.2	POWER	10 WATTS MAXIMUM
9.3	STARTING TIME	20 SECONDS NOMINAL
10.0	INSULATION RESISTANCE	50 MEGOHMS BETWEEN EACH LEAD AND CASE
		WITH 200 VDC APPLIED. MOTOR LEADS
		EXEMPT
•11.0	SEALING	HERMETIC, EPOXY SEALED FOR 0 TO 20,000
		FEET ALTITUDE ENVIRONMENT
•12.0	FINISH	ANODIZED ALUMINUM
13.0	SERVICE LIFE	500 HOURS ON MOTOR OR 1.0 X 10⁶ CYCLES
		ON POT.
14.0	WEIGHT	14 OUNCES APPROXIMATELY
15.0	OVER-RANGE	200% OF ACTUAL RATE
16.0	ENVIRONMENTAL CONDITIONS	
16.1	TEMPERATURE	-65°F TO 165°F
16.2	SHOCK	50G, 11 ± 1 MS DURATION
16.3	ACCELERATION	20G STEADY STATE
16.4	VIBRATION	0.06 D.A. OR 10G, WHICHEVER IS LIMITING,
		20 TO 500 Hz
•17.0	EQUIVALENT CONTACT RESISTANCE	200 OHMS MAXIMUM
18.0	REMARKS	
18.1	ITEMS MARKED WITH (•) ARE CHECKED IN PRODUCTION TESTS. OTHER ITEMS FOR
	REFERENCE MAY BE CHECKED ON ORDER BY QUALIFICATION TEST.

CONTROLLED DRAWING

Humphrey inc
SAN DIEGO CALIFORNIA

ELECTRO-MECHANICAL
INSTRUMENTS

TITLE
ENVELOPE DRAWING
GYROSCOPE-RATE

SCALE 1/1

RG51-0106-1 E

CODE IDENT NO 09084

VIEW A-A

RECEPTACLE (CANNON
#DEM-9P-001

NAMEPLATE, .010
MAX THICKNESS

4 MOUNTING HOLES

RATE INPUT
AXIS MARKER

.35 MAX

3.45±.06

WIPER MOVEMENT FOR
CLOCKWISE ROTATION ABOUT
INPUT AXIS MARKER

NOISE FILTER

2 3 4 5 6 7 8 9

MAGNETIC FLUX DETECTOR

DIRECTIONAL GYROSCOPE

FLUX ALIGNMENT TORQUER

AMPLIFIER

HIGH SPEED NORTH SLAVING

D.C. SUPPLY

INNER GIMBAL TORQUER SYSTEM

S3 S1
S2
R1
R2

E H D B A
A B C D E
A B C D E

A
B
C
D
E

F
G

H

J
L
M
N
P
R
S
T
U

SLAVING INDICATOR

115 VAC, 400 Hz, 1PH

COMMON

YAW SYNCHRO OUTPUT

COMMON

26 VAC, 400 Hz, 1 PH

SPARES

ELECTRO-MECHANICAL INSTRUMENTS

TITLE ENVELOPE DRAWING
DIRECTIONAL GYRO & FLUX GATE SYSTEM

UNLESS OTHERWISE SPECIFIED LIMITS ON DIMENSIONS ARE
DECIMAL .XX = ± .01
.XXX = ± .010
ANGULAR = ± ½°

SCALE 1/1
DWG NO DG11-O105-1
SIZE C
REV F

SHEET 1 OF 2

CODE IDENT NO. 99284

CUSTOMER REF MATL

REVISIONS

45°

120° (3 PL.)

THIS END UP

FWD

FWD

RECEPTACLE - BENDIX
#PT07H-14-18P
(REF) MATING PLUG
#PT06A-14-18S
FURNISHED

YAW A216 MARKER

.168 + .004 DIA THRU.
4 HOLES EQUALLY SPACED ON A 3.500 DIA.

.156 X .50
3 SLOTS EQ.
SPACED ON A 2.91 DIA B.C.

3.77 DIA MAX.

8.25 MAX.

.81 MAX.

NAMEPLATE

RECEPTACLE - WINCHESTER
#MSP-LRN
(REF) MATING PLUG
#MSS-LSH19C FURNISHED

FLUX DETECTOR
FD01-O101-1

INPUT
OUTPUT
H E B A D

1.91 MAX.

.12

2.28 DIA MAX.
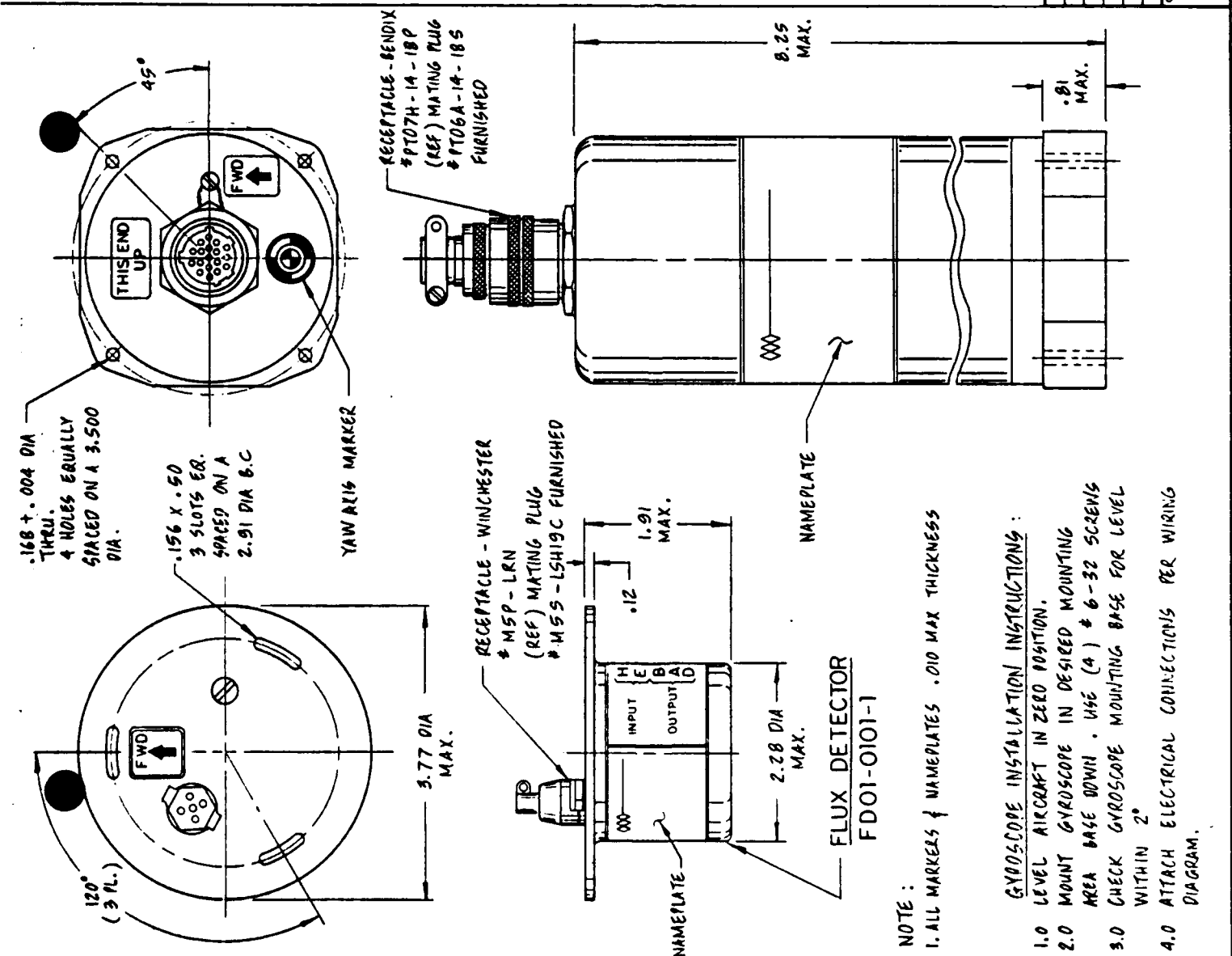
NAMEPLATE
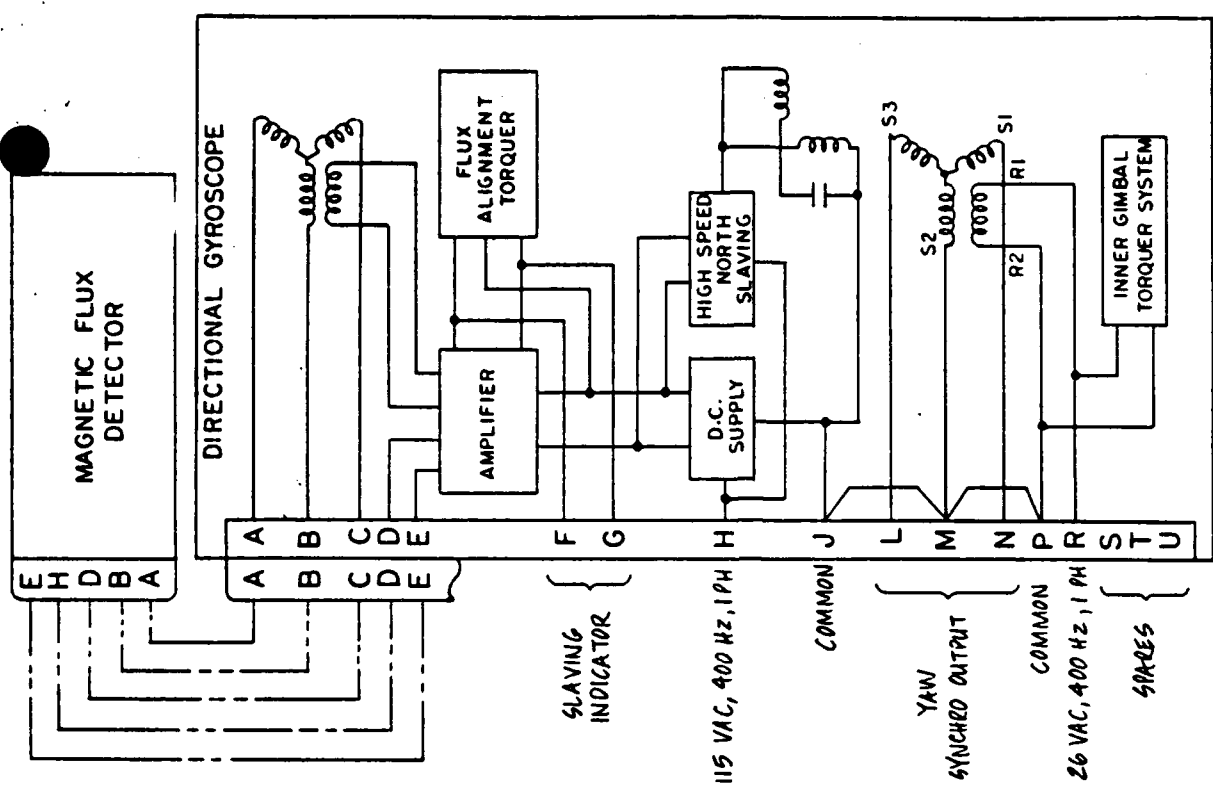
NOTE:
1. ALL MARKERS & NAMEPLATES .010 MAX THICKNESS

GYROSCOPE INSTALLATION INSTRUCTIONS:

1.0 LEVEL AIRCRAFT IN ZERO POSITION.
2.0 MOUNT GYROSCOPE IN DESIRED MOUNTING AREA BASE DOWN. USE (4) # 6-32 SCREWS
3.0 CHECK GYROSCOPE MOUNTING BASE FOR LEVEL WITHIN 2°
4.0 ATTACH ELECTRICAL CONNECTIONS PER WIRING DIAGRAM.

SPECIFICATIONS

*1.0	RANGE	±10°/SECOND
2.0	DAMPING	0.7 ±0.2
3.0	FREQUENCY RESPONSE	5.0 HZ FOR 90° PHASE LAG
4.0	INPUT	
4.1	VOLTAGE	±15 VOLTS
4.2	CURRENT	100 MILLIAMPS
5.0	OUTPUT	
*5.1	FULL SCALE	0 TO 5.0 VDC ±10%
*5.2	SCALE FACTOR	0.25 V/°/SEC. NOMINAL
*5.3	NULL	±2.5 VDC ±300 MILLIVOLT NOMINAL
*5.4	LINEARITY	±1 OF FULL SCALE
*5.5	NOISE	100 MILLIVOLTS RMS TYPICAL
*5.6	LINEAR ACCEL. SENS.	0.1% OF FULL SCALE/G
5.7	THRESHOLD	0.01 DEG/SEC.
5.8	HYSTERESIS	ESSENTIALLY NONE
5.9	RESOLUTION	INFINITE
5.10	CROSS AXIS SENSITIVITY	0.02°/SEC/°/SEC. INPUT
6.0	SEALING	HERMETIC
7.0	LIFE	10,000 HOURS MINIMUM
8.0	WEIGHT	12 OUNCES NOMINAL
9.0	ENVIRONMENTAL CONDITIONS	
9.1	TEMPERATURE RANGE	-55 TO +70 DEG. C
9.2	SHOCK	100G, 11 ±1 MILLISECOND
9.3	VIBRATION	0.06 INCH D.A. OR 100G WHICHEVER LIMITS, 5 TO 2000 HZ
9.4	ACCELERATION	100G ANY AXIS
9.5	HUMIDITY	100%
9.6	ALTITUDE	UNLIMITED
10.0	REMARKS	
10.1	ITEMS MARKED WITH (*) ARE CHECKED IN PRODUCTION TESTS. OTHER ITEMS FOR REFERENCE MAY BE CHECKED ON ORDER BY QUALIFICATION TEST.	
10.2	BECAUSE OF THE LOW RATE-RANGE AND HIGH AMPLIFICATION OF THE SIGNAL, NULL STABILITY WITH RUNNING TIME AND/OR TEMPERATURE CHANGE MAY SEEM EXCESSIVE. ALLOW ABOUT 30 MINUTES OPERATING TIME FOR NULL TO STABALIZE. REMEMBER, OUTPUT IS ACCURATE WHEN REFERENCED TO EXISTING NULL.	
10.3	THIS UNIT CONTAINS NO MOVING PARTS AND IS NOT DAMAGED BY EXCESSIVE RATE INPUTS.	

CONTROLLED DRAWING

ELECTRO-MECHANICAL INSTRUMENTS

Humphrey

TITLE
ENVELOPE DRAWING
RATE TRANSDUCER

UNLESS OTHERWISE SPECIFIED
LIMITS ON DIMENSIONS ARE

DRAWN BY SCHWARTZ
MODEL NO. RT03-0153-1
CODE IDENT NO 94204
DWG C SIZE
SCALE NONE

REVISIONS (SEE DCN)

2.125 (53.98)
.180 +.005 -.003 (4.57) DIA THRU 4 MTG HOLES
2.605 (66.17)
2.125 (53.98)
2.605 (66.17)

CONNECTOR - BENDIX PT02H-12-8P, MATES WITH PT06E-12-8S(SR) (CONNECTOR ORIENTATION OPTIONAL)

AXIS MARKER

NOTE: DIMENSIONS IN PARENTHESES ARE IN MILLIMETERS.

2.56 DIA (65.02)
2.31 DIA (58.67)
.50 MAX (12.7)
3.80 (96.52)
.40 (10.16)
.20 (5.08)

CONNECTOR

LOW RATE OUTPUT	F	
HIGH RATE OUTPUT (ALTERNATE)	E	
SIGNAL COMMON	D	
COMMON	C	
+15 VDC	B	
-15 VDC	A	
CASE GROUND	G	

OSC
PUMP
OSC
AMP
AMP
REGULATORS

Appendix D -   **Calibration data**

MODEL NO. VG24-1904-1   TECHNICIAN _FB_   DATE _8-1-88_

SERIAL NO. _123_ (🐝)   INSPECTOR _Philisola_   DATE _8-3-88_

SPECIFICATION PARAGRAPH	TEST DATA	TEST REQUIRED
4.3 Erection Rates	Pitch _97_ Sec.	90 - 300 sec. (2 - 6.7°/min.)
	Roll _104_ Sec.	90 - 300 sec. (2 - 6.7°/min.)
3.2 Starting Current	_0.48_ Amps	2.5 amps max.
3.2.1 Running Current	_0.25_ Amps	.7 amp max.
4.4 Vertical Accuracy	Pitch _501_ VR	.487 - .513 VR
	Roll _500_ VR	.490 - .510 VR
1.1 Mechanical Travel	Pitch _✓_ Acceptable	$\pm 60°$ min.
	Roll _✓_ Acceptable	360° continuous
1.2 Electrical Travel	Pitch $\pm 55°$	$\pm 55°$, $\pm 2.5°$
	Roll $\pm 70°$	$\pm 70°$, $\pm 2.5°$
4.2 Time to Erect From Motor Off	_25_ Sec.	Less than 480 sec. (8 min.)
Potentiometer	Pitch _✓_ Acceptable	Resistance decreases
Polarity	Roll _✓_ Acceptable	Resistance decreases
5.15 Insulation Resistance	> 25 Megohms	Greater than 25 megohms at 50 VDC
5.18 Sealing	_✓_ Acceptable	Vacuum equivalent to 25,000 feet.

SPECIFICATION
PARAGRAPH   TEST DATA   TEST REQUIRED

2.1   Static Error Band

(Linearity)

Pitch Angle	Voltage Ratio	Acceptable Range VR
55°	999	.980 – 1.000
50°	953	.935 – .973
40°	862	.847 – .881
30°	771	.758 – .788
20°	678	.668 – .696
10°	590	.579 – .603
0°	501	.489 – .511
10°	412	.397 – .421
20°	319	.304 – .332
30°	229	.212 – .242
40°	139	.119 – .153
50°	047	.027 – .065
55°	003	.000 – .020

Roll Angle		
70°	997	.986 – 1.000
60°	925	.915 – .943
40°	783	.774 – .798
20°	641	.633 – .653
0°	500	.492 – .508
20°	357	.347 – .367
40°	212	.202 – .226
60°	069	.057 – .085
70°	002	.000 – .016

2.3   Contact Resistance   Pitch < 100 Ohms

Roll < 100 Ohms

2.2   Potentiometer
Resistance   Pitch 1,999 Ohms   1,900 – 2,100 Ohms

Roll 2,028 Ohms   1,900 – 2,100 Ohms

Appendix D -   Mounting material

E·A·R

C-1002 damped isolation materials

C-1002 offers performance, versatility for wide range of applications

While E-A-R's ISODAMP® C-1002 thermoplastic originally was developed for use in constrained-layer damping systems, the versatile, high-performance damping and isolation material today has more diverse applications.

With its high material-loss factor and excellent physical properties, C-1002 effectively controls noise, vibration, shock and motion in applications ranging from sensitive medical equipment to military ordnance vehicles. It is available as sheets, rolls, die-cut parts and standard and custom injection-molded parts, as well as in special, temperature-tuned formulations.

Figure
Comparison-Transmissibility.

C-1002 is soft and flexible, yet physically strong and resistant to tearing, abrasion and skid. It exhibits low rebound from impact, high dielectric strength, and excellent flammability properties, complying with UL 94V-0, FAR and FMVSS302. C-1002 also resists degradation from ozone, UV radiation, fungus, bacteria and chemicals.

Because it combines superior damping performance and strength with weight efficiency, C-1002 thermoplastic is widely used in constrained-layer damping (CLD) systems to reduce noise and vibration in weight-sensitive applications, such as aircraft, ships

and off-road vehicles. C-1002 is found in CLD systems incorporating materials ranging from common plywood to multi-layer composite laminates.

To control intense vibrational disturbances in weight-sensitive applications, C-1002 is used as the constraining layer in multi-damping layer (MDL) composites with one of E-A-R's C-3000 Series energy absorbing foams as a base layer. MDL composites are used in a variety of high-performance vehicle applications such as aircraft fuselage prop plane and engine mount areas.

Improved system performance

As equipment mounts, feet and isolators, C-1002 protects against broad band and variable force excitation by damping system resonances. In comparison, materials such as traditional rubber greatly amplify the natural frequency resonance and all higher order resonances, thus reducing the amount of isolation possible.

As precision-molded parts, such as grommets, crash stops and inertial dampers, C-1002 is used extensively in office equipment, computer disk drives and other compact electromechanical systems. It improves operational speed and accuracy by damping and isolating resonances and rapidly dissipating shock energy.

In addition, highly damped C-1002 provides controlled deceleration after initial shock input. More lightly damped materials not only amplify the initial shock input, but also expose the system to significant and prolonged aftershock motions.

For more information, see E-A-R's General Catalog No. 502, the ISODAMP C-1000 Series Grommets and Custom Parts Catalog No. 716 and technical data sheet MDS-19 (ISODAMP C-1000 Series Thermoplastics). For additional assistance, contact an E-A-R noise control expert at (317) 872-1111.

Variety of materials adapt for many uses

Whether an application requires a quarter-inch-thick damping sheet or a custom-engineered isolation mount, there is an E-A-R C-1002 product form to fit the bill.

In sheet and roll form, C-1002 is available in thicknesses from .015-inch to 1 inch. C-1002 sheet and roll materials can be combined with other E-A-R materials to form multifunction composites that provide weight-efficient high performance. Its excellent physical properties and wear resistance make C-1002 well-suited for constrained-layer damping applications in harsh environments.

E-A-R offers 41 standard C-1002 isolators in 15 different styles, as well as custom die-cut and injection-molded parts. Because injection molding with proprietary ISODAMP thermoplastic involves shorter cycle times and less waste than processing neoprene, silicone or thermosetting rubbers, small-volume runs and prototype parts are more cost-effective.

C-1002 materials can be die-, shear- or knife-cut and are adhesively bonded to clean, degreased substrates. Pressure-sensitive adhesive backings are available for convenient installation. For further ease of assembly, E-A-R can ship parts in one-application kits.
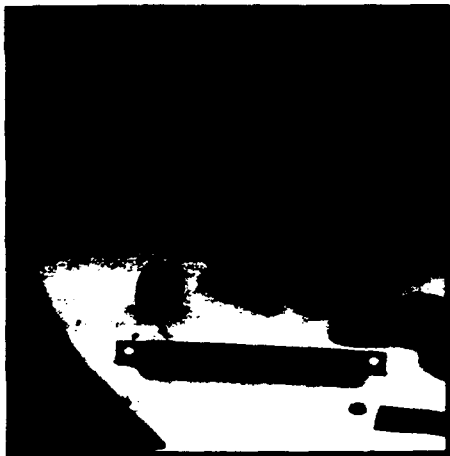

ISODAMP C-1002 materials.

ISODAMP® C-1000 SERIES THERMOPLASTICS

Description

E-A-R ISODAMP C-1000 Series thermo-plastics are high-performance vibration isolation/damping materials. Composed of energy absorbing thermoplastic alloys, the material series is typified by extra-ordinarily high material loss factors. The high internal damping of ISODAMP Series materials reduces mechanically or acous-tically induced vibrations and dissipates shock and impact energy at a very rapid rate. These properties—in conjunction with physical strength, flexibility, environ-mental resistance, anti-skid properties and good flame resistance—make C-1000 Series materials excellent for constrained layer damping, damped isolation and impact control applications.

Available in three peak-performance temperature-tuned formulations, ISODAMP C-1000 Series materials pro-vide maximum energy control in a wide range of applications.

ISODAMP Formulation	Peak Damping Performance Temperature Range (°F)	Shore A Durometer*
C-1002	55-105	56
C-1105	80-130	63
C-1100	95-145	70

*Shore A Durometer (15-sec. test)

ISODAMP C-1002 and C-1100 are avail-able in sheets, rolls or as custom die-cut parts. And, ISODAMP C-1002, C-1105 and C-1100 are available in standard and custom injection-molded parts.

Applications

Constrained Layer Damping

ISODAMP C-1000 Series thermoplastics originally were developed for use as high-performance constrained layer damping materials. ISODAMP C-1002 is widely used to reduce mechanically induced vibrational disturbances and noise in air-craft, military and non-military ships, off-road vehicles, continuous miners, office equipment, and articulated and non-articulated assembly equipment. The materials are used in a variety of con-strained layer systems—from common plywood, aluminum and steel to state-of-the-art weight-sensitive honeycomb composite structures.

Constraining Layer for MDL

E-A-R C-1002 is a primary building block for the unique E-A-R ISODAMP MDL (multi-damping layer) damping composites. MDL composites provide broad temperature damping in weight-sensitive applications where maximum energy control is required. MDL composites are primarily used by the aircraft and aerospace indus-tries as well as in other high-performance vehicle applications (reference MDS-50). The composites are widely used to provide weight-efficient control of high-intensity acoustical and mechanical energy in areas such as crossover ducts, surfaces in close proximity to prop planes or in engine mounting areas.

Isolation Mounts and Equipment Feet

E-A-R C-1000 Series materials are mate-rials of choice for equipment mounts, pads and feet in both OEM original design and aftermarket applications. In addition to vibration isolation, C-1000 Series mounts provide protection from broad-band and variable forcing frequency inputs by damping amplification at system resonances and by dissipating vibrational energy from the system. E-A-R provides standard mounts, custom-molded parts and sheet material for do-it-yourself installation.

MATERIAL SOLUTIONS FOR ENERGY CONTROL
NOISE, VIBRATION, SHOCK, MOTION

Grommets, Bushings, Crash Stops

ISODAMP C-1000 Series thermoplastics are extensively used in office machines and computer peripherals as well as in electromechanical equipment of all types to control motion, vibration, shock and noise. In hard disk drives and similar sensitive precision equipment, C-1000 Series grommets produce a more dynamically compliant system. In systems where space is at a premium, C-1000 Series grommets and bushings quickly dissipate shock energy and restore static equilibrium without oscillation or large initial displacements. In mechanical systems such as copiers and printers, C-1000 Series parts are used as motor mounts, printer platen and frame/case isolators. Crash stops made of C-1000 Series materials can reduce cycle time and increase reliability by providing controlled impact deceleration with little or no rebound. See E-A-R Grommet Catalog Form 714 for more information on the standard ISODAMP grommet and bushing product line.

Ordnance Vehicle Applications

Due to their compression-velocity sensitivity and unique energy absorbing and dissipation characteristics, ISODAMP C-1000 Series materials are well-suited for many specialty ordnance vehicle applications. The M1A1 tank capitalizes on ISODAMP's excellent physical integrity and the diversity of available material forms, incorporating ISODAMP C-1002 die-cut sheets and custom-molded fratricide bars to provide enhanced protection for the ammunition storage system, reducing the overall vulnerability of the current production main battle tank. In addition, other potential applications include interior spall protection mechanisms, shielding, and composite and ceramic armor systems.
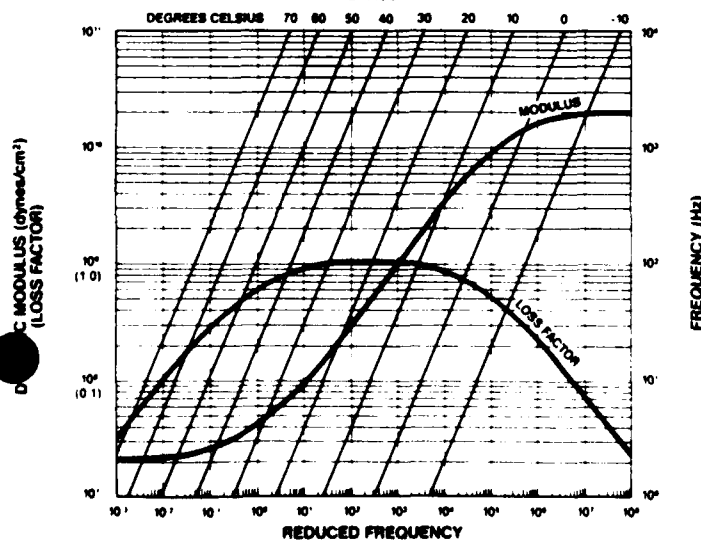
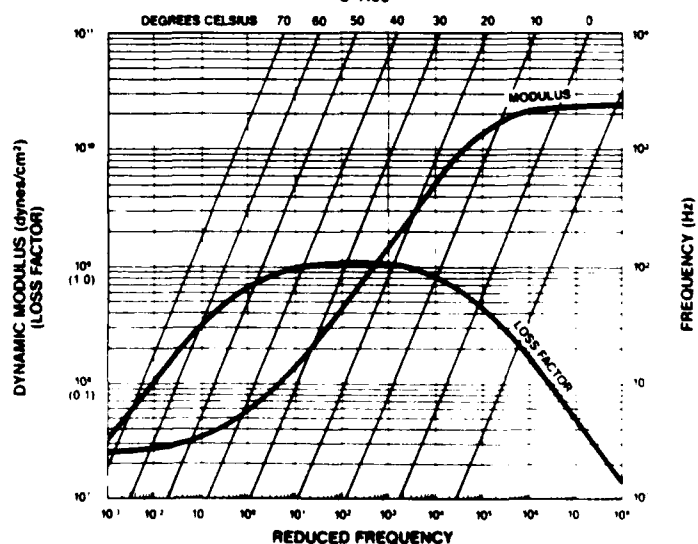# DAMPING CHARACTERISTICS

C-1002

Instructions—Reduced Frequency Nomograms

The reduced frequency format is the standard method for displaying damping material performance data. To determine dynamic Young's modulus and material loss factor at a given temperature and frequency, use the following steps: 1) Select the frequency of interest on the right-hand vertical axis. 2) Follow the selected frequency line horizontally to the left until the selected *diagonal* temperature isotherm is intersected. 3) Draw a vertical line up and down through the frequency/isotherm intersection (this vertical line will intersect the dynamic Young's modulus and material loss factor curves). 4) Draw horizontal lines from these points to intersect the left-hand vertical axis. 5) The dynamic Young's modulus value is read using the Dynamic Modulus scale and the loss factor from the (Loss Factor) scale.

C-1105

C-1100

TYPICAL PROPERTIES

Property	Test Method	C-1002	C-1105	C-1100
Physical Properties				
Specific Gravity	ASTM D792	1.289	1.287	1.282
Glass Transition, T_g	ASTM E756	−20°C	−13°C	2°C
Hardness	ASTM D2240 Shore A durometer 15 sec. post impact @ 23°C	56	63	70
Rebound	ASTM D2632 (Modified) Bashore Resilience % Rebound (First) Min. Rebound Temp.	4.8% 21°C	5.4% 34°C	5.7% 40°C
Outgassing	ASTM E595 (Modified) 24 hr. at 10^{-6} Torr Total Mass Loss Water Reabsorbed	0.067% @ 40° 0.043%	Not Tested	0.135% @ 50°C 0.045%
Dielectric Strength	ASTM D149 Breakdown Voltage	166 volts/mil	Not Tested	Not Tested
Thermal Conductivity	ASTM C177 BTU in./hr. ft.² °F	1.00	Not Tested	.90
Coefficient of Friction	ASTM D3389 on Etched Aluminum Static Kinetic	.92 .75	1.21 .77	1.24 .71
Flammability	UL 94 Vertical 0.125" Samples FAR 25.855 (a-1) FAR 25.853 (b) FAR 25.853 (b-3) FMVSS-302	V-0 Recognized V-0 to 0.056" Meets at 0.060" Meets at 0.030" Meets at 0.015" Meets at 0.015"	V-0 Not Applicable	V-0 Meets at 0.060" Meets at 0.060" Meets at 0.060" Meets at 0.060"
Strength Properties				
Compressive Deformation	ASTM D621 Method B 24°C % Deformation (3 hr.) % Recovery (1.5 hr.)	10.4% 90.4%	9.8% 95.5%	8.4% 95.7%
Compression Set	ASTM D395 Method B 22 hr. at 22°C (72°F) 22 hr. at 80°C (176°F)	14% 62%	23% 51%	24% 55%
Tensile Strength	ASTM D903	1574 psi	1807 psi	2058 psi
Elongation	ASTM D903	459%	417%	317%
Tensile Modulus	ASTM D903	450 psi	805 psi	1155 psi
Tear Strength	ASTM D1004 0.125" Samples	25.2 lb.	30.1 lb.	38.1 lb.
Abrasion Resistance	ASTM D3389 H22 stone, 1000g load Wear Factor	242	350	271
Environmental Resistance				
Ozone Resistance	ASTM D1149	Not Affected	Not Affected	Not Affected
Ultraviolet Resistance	ASTM G84 (300 hr.)	Not Affected	Not Affected	Not Affected
Accelerated Aging	ASTM G23 Method 1 Weather-Ometer 1000 hr. Exposure to Cycles 102 min. Light (carbon arc) @ 50% RH & 63°C, 18 min. Light and Water Spray	Decrease in Gloss, No Other Significant Effects Noted	Not Tested	Not Tested
Bacterial Resistance	ASTM G22	Resistant No Growth	Not Tested	Not Tested
Fungal Resistance	ASTM G21	Resistant No Growth	Not Tested	Not Tested
Chemical Resistance	ASTM D543 1 wk. Immersion % Weight Change in Reagent: Sulfuric Acid (2 molar) Diesel Fuel Distilled Water Sea Water Mineral Oil Ethylene Glycol	0.00% +2.91% +0.36% +0.36% −0.36% −0.36%	+0.38% +2.62% +0.36% +0.37% −0.38% +1.16%	+0.39% +0.92% +0.39% +0.39% 0.00% 0.00%
Temperature Range	Peak Damping Performance Range	55 to 105°F	80 to 130°F	95 to 145 °F
	Recommended Maximum Intermittent	180°F	180°F	180 F

ISODAMP C-1000 SERIES THERMOPLASTICS

Specifications
Isolation/constrained layer damping materials shall be E-A-R ISODAMP C-1000 Series thermoplastics manufactured by E-A-R Division, Cabot Corporation, Indianapolis, Indiana.

C-1002-01	0.015" thick
C-1002-03	0.03" thick
C-1002-06	0.06" thick
C-1002-12	0.12" thick
C-1002-25	0.25" thick
C-1002-50	0.50" thick
C-1002-99	1.00" thick
C-1002	molded parts
C-1105	molded parts
C-1100-06	0.06" thick
C-1100-12	0.12" thick
C-1100	molded parts

E-A-R ISODAMP C-1000 Series thermoplastics can be die, shear or knife-cut. E-A-R molds a broad line of standard vibration, shock and motion control parts and offers custom molding and die cutting.

The ISODAMP C-1000 Series materials ay be cryogenically machined to pro- ce prototype parts. Cooling by continually applying a Venturi-effect air gun to the cutting tool or freezing ISODAMP prior to machining will sufficiently harden the material for machining.

ISODAMP C-1000 Series materials can be bonded to clean, degreased substrates with Bostik 7132/Boscodur 4 adhesive system. Refer to E-A-R MDS-25A for detailed properties.

For design and application recommendations, contact your nearest E-A-R representative or sales office.

Velocity-Sensitive Compression Resistance

E-A-R ISODAMP C-1000 Series materials are highly compression-velocity sensitive. This sensitivity is similar to that displayed by viscous dampers. Like a shock absorber, if C-1000 Series ma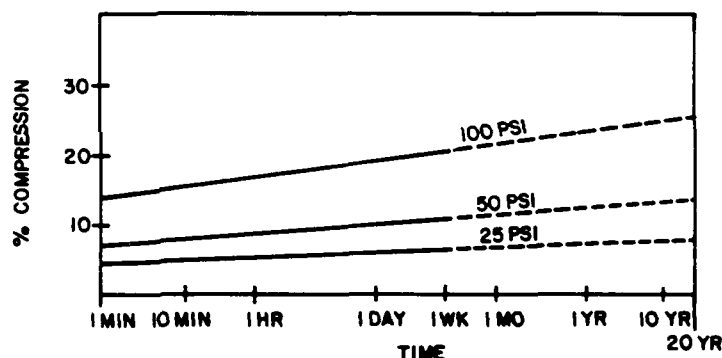terials are compressed quickly, they appear stiff; if they are compressed slowly, they appear soft. Compression-velocity sensitivity is one of the keys to ISODAMP C-1000 Series' excellent shock absorbing and low rebound properties.

Compressive Creep

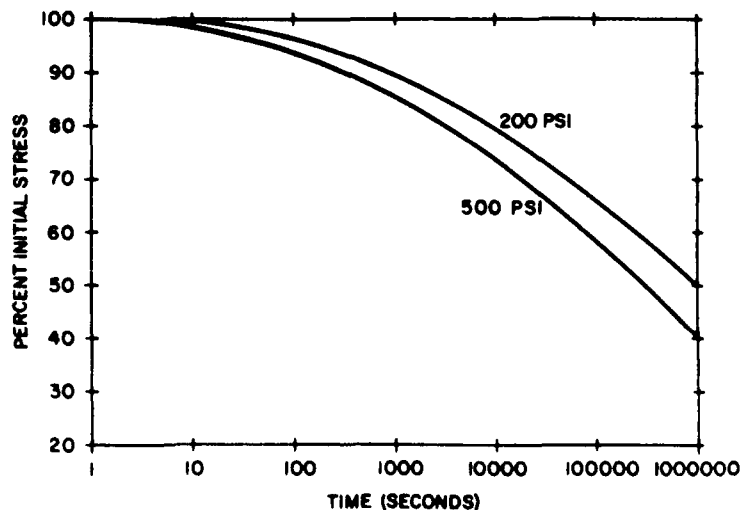In isolation systems C-1000 Series materials are recommended for a 50 psi optimum load. The compressive creep curves for 25, 50 and 100 psi are shown. The data represent C-1002 at 70°F with a shape factor of 0.5.

Stress Relaxation

ISODAMP C-1000 Series materials are often used in gasket or washer applications. Stress relaxation data for 30 mil C-1002 per ASTM F-38 conducted at 72°F are shown.
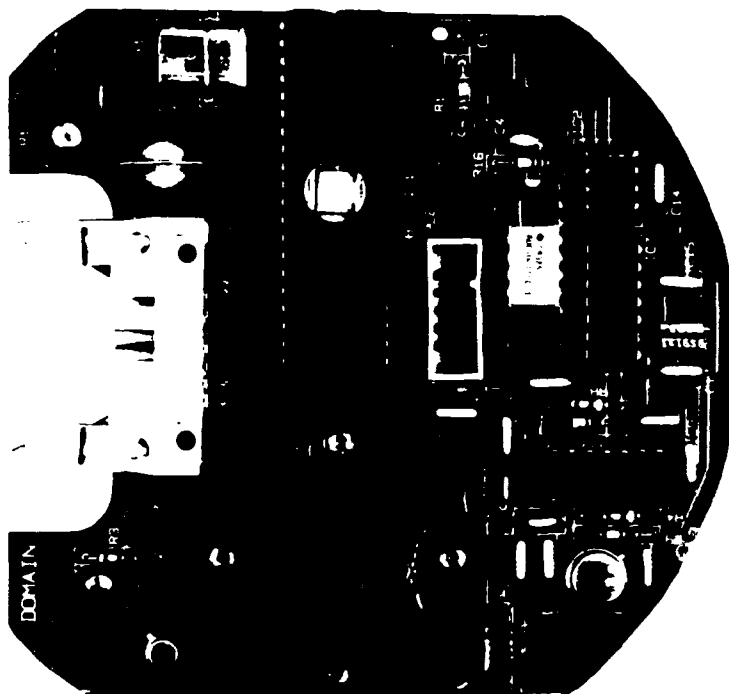
Appendix E - **Three-Axis Magnetometer Data Sheets**
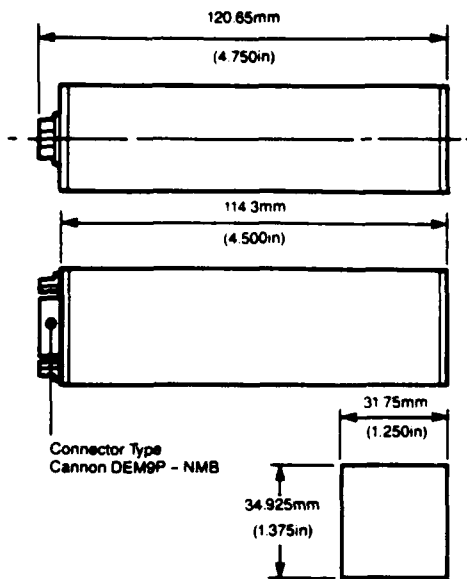
For HEADING and ATTITUDE SENSING APPLICATIONS

Magnetometers are fast becoming the accepted device to supply either Heading and or Attitude information.

They are small, light and cost effective and may be used in a range of operational environments either in the air on and or under water.

As a solid state advance technology device they offer versatility to the system designer and can operate with either digital or analog interfacing.

TAM/7

Our new development is a self contained hybrid strapdown magnetic field measuring module having three axes of measurement. The package consists of fluxgate field sensors and hybrid circuits plus some discrete components mounted on a PCB. All are contained within an extruded aluminum case.
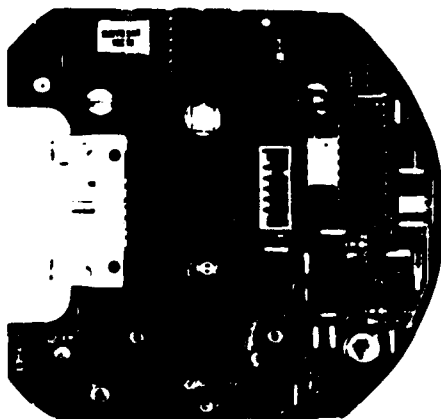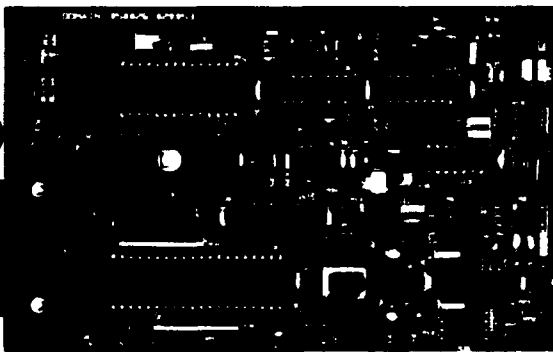
Typical uses are:

Fixed and rotary wing aircraft; unmanned aircraft; underwater remotely operated vehicles; decoys; missiles; soundings rockets and satellites; targets; torpedoes; current meters;

120 65mm
(4 750in)

114 3mm
(4 500in)

31 75mm
(1.250in)

34.925mm
(1.375in)

Connector Type
Cannon DEM9P - NMB

8/10 BIT HEADING SYSTEM DHS 2

A two axis strapdown fluxgate compass giving heading angle output as a digital word (8, 9 or 10 bit). Secondary outputs in analog form of d.c. voltages proportional to Sin θ and Cos θ are provided as well as a scaled d.c. 0 to 3.6V signal proportional to 0° to 360°

The compass system has no gravitational reference as the two axis fluxgate sensor is not normally gimballed. Systems having gimballed sensors have been provided, however, in the past.

Typical uses are: Sonobuoys; RPV's/Drones; Air and Sea Missiles; Databuoys and Navigation Systems.

ANALOG COMPASS SYSTEM

A simple two axis analog fluxgate compass giving output signals of Sin θ and Cos θ The magnetometer also has auto gain circuitry to allow for use anywhere. irrespective of latitude.
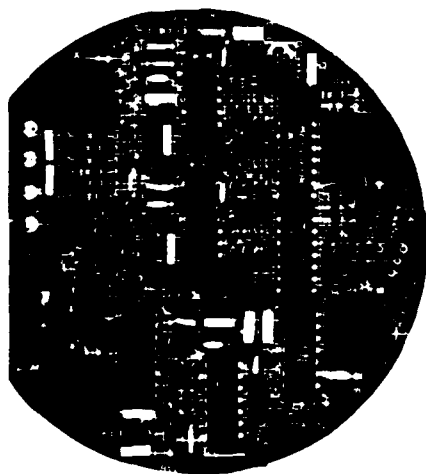
Typical uses are: Sonobuoys. Databuoys and simple Navigation Systems.

AHRS MAGNETOMETERS TYPE TAM/4/6

A three axis magnetometer used in Attitude and Heading Reference Systems (AHRS). Outputs are in the form of three channels of analog processed signals. The AHRS can be corrected for orthogonality and are available with uni-polar or bi-polar power supplies. These units will be of MIL-Spec design and contain appropriate circuits and components.

Typical uses are: As part of fully attitude compensated AHRS for prime or standby use in Aircraft (fixed or rotary wing) or RPV's/Drones; in Air or Sea Missiles; in Arrays at Sea; in Land-based Measuring Instruments; and in Navigation Systems.

DIGITAL HEADING SENSOR DHS 3

A strapdown fluxgate compass having its two axis sensor integral with its electronics board. Heading angle output is in the form of a digital word (8, 9 or 10 bit).

Typical uses are: Sonobuoys; Databuoys; RPV's/Drones; and Navigational Systems.

SENSORS

Most sensors are preferred to be provided as strapdown devices although gimballed versions are available. Sensors can be provided as single, double or triaxial fluxgates.

The fluxgate sensor is a wound component with a toroidal ferromagnetic core. A drive field is applied to the core and the external field interaction with it produces an assymetric change of core flux. This changing core flux is detected by a solenoidal winding over the core and the resultant signal is processed.
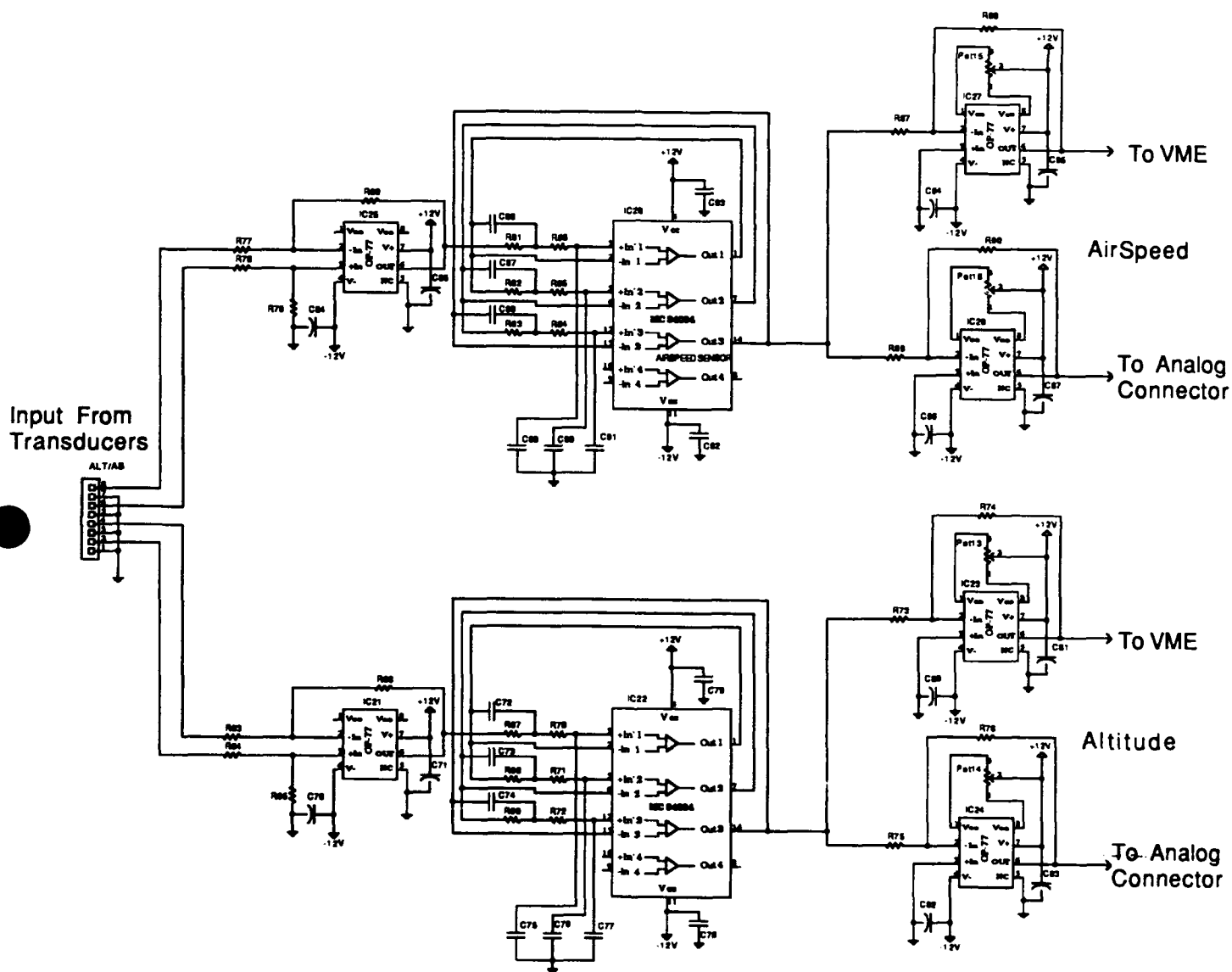
Individual sensor packages vary in size dependent upon the trade-offs accepted against performance, cost, power etc.

3rd ANGLE PROJECTION

SHEET No.1 OF 1

100 MILLIMETRES FULL SIZE

0   50   100

DOMAIN TYPE & S/S 050130

1

3

4   TO BE PLACED AFTER · FITTING K3

TO BE PLACED AFTER FITTING K4.

2

MATERIAL		SPECIFICATION	ESTIMATED MASS	SCALE 2:1	TITLE	RESPONSIBLE AUTHORITY
					TAM 4 PCB ASSEMBLY	SECURITY CLASSIFICATION
FINISH		SPECIFICATION	TOLS	DRAWN IN	N.S.N	DOMAIN MAGNETICS DRG No 3-020167
					DOWTY ELECTRONIC SYSTEMS LTD DOMAIN MAGNETICS	
					MOD ESTABLISHMENT	SERVICE DRG No
ISSUE	DATE					SHEET No.1 OF 1

Appendix F -   Signal Conditioning Board

Appendix F -   **PCB Schematics**

Input From
Transducers

To VME

AirSpeed

To Analog
Connector

To VME

Altitude

To Analog
Connector

Input
From
Display
Board

To VME

Roll Out

To Analog
Connector

To VME

Pitch Out

To Analog
Connector

BCM Designs

FIP Main Board
Page 2 of 3
Revision 1.0
Completion Date : Oct. 9, 1989

Input
From
3-Axis
Magno-
meter

3-Axis

To VME

Z-Axis

To Analog
Connector

To VME

Y-Axis

To Analog
Connector

To VME

X-Axis

To Analog
Connector

Appendix F - **PCB layouts**

Appendix F -   **Parts Data Sheets**

OP-77
NEXT GENERATION OP-07
(ULTRA-LOW OFFSET VOLTAGE OPERATIONAL AMPLIFIER)

Precision Monolithics Inc.

PRELIMINARY

FEATURES

- Outstanding Gain Linearity
- Ultra High Gain .... 5000V/mV Min
- Low V_{OS} .... 25μV Max
- Excellent TCV_{OS} .... 0.3μV/°C Max
- High PSRR .... 3μV/V Max
- High CMRR .... 1.0μV/V Max
- Low Power Consumption .... 60mW Max
- Fits OP-07, 725, 108A/308A, 741 Sockets

ORDERING INFORMATION†

	PACKAGE		
TO-99 8-PIN	HERMETIC DIP 8-PIN	PLASTIC DIP 8-PIN	OPERATING TEMPERATURE RANGE
OP77AJ*	OP77AZ*		MIL
OP77EJ	OP77EZ	OP77EP	IND
OP77BJ*	OP77BZ*		MIL
OP77FJ	OP77FZ	OP77FP	IND
	OP77GZ	OP77GP	COM

*For devices processed in total compliance to MIL-STD-883, add /883 after part number. Consult factory for 883 data sheet.

†All commercial and industrial temperature range parts are available with burn-in. For ordering information see 1986 Data Book, Section 2.

GENERAL DESCRIPTION

The OP-77 significantly advances the state-of-the-art in precision op amps. The OP-77's outstanding gain of 10,000,000 or more is maintained over the full ±10V output range. This exceptional gain-linearity eliminates incorrectable system nonlinearities common in previous monolithic op amps, and provides superior performance in high closed-loop-gain applications.

Low initial V_{OS} drift and rapid stabilization time, combined with only 50mW power consumption, are significant improvements over previous designs. These characteristics, plus the exceptional TCV_{OS} of 0.3μV/°C maximum and the low V_{OS} of 25μV maximum, eliminates the need for V_{OS} adjustment and increases system accuracy over temperature.

PSRR of 3μV/V (110dB) and CMRR of 1.0μV/V maximum virtually eliminate errors caused by power supply drifts and common-mode signals. This combination of outstanding characteristics makes the OP-77 ideally suited for high-resolution instrumentation and other tight error budget systems.

This product is available in five standard grades and three standard packages: the TO-99 can and the 8-pin mini-dip in ceramic or epoxy.

The OP-77 is a direct or upgrade replacement for the OP-07, OP-05, 725, or 108A op amps. 741-types can be replaced by eliminating the V_{OS} adjust pot.

PIN CONNECTIONS

TO-99 (J-Suffix)

EPOXY MINI-DIP (P-Suffix)
8-PIN HERMETIC DIP
(Z-Suffix)

SIMPLIFIED SCHEMATIC

1/86, Rev. A

OPERATIONAL AMPLIFIERS

ABSOLUTE MAXIMUM RATINGS (Note 2)

Supply Voltage . . . . ±22V
Internal Power Dissipation (Note 1) 500mW
Differential Input Voltage . . . . ±30V
Input Voltage (Note 3) . . . . ±22V
Output Short-Circuit Duration . . . . .Indefinite
Storage Temperature Range
  J and Z Packages . . . . −65°C to +150°C
  P Package . . . . −65°C to +125°C
Operating Temperature Range
  OP-77A, OP-77B . . . . −55°C to +125°C
  OP-77E, OP-77F . . . . −25°C to +85°C
  OP-77G . . . . 0°C to 70°C
Lead Temperature Range (Soldering, 60 sec) . . . . . . .300°C
DICE Junction Temperature (T_j) . . . . −65°C to +150°C

NOTES:
1. See table for maximum ambient temperature rating and derating factor

PACKAGE TYPE	MAXIMUM AMBIENT TEMPERATURE FOR RATING	DERATE ABOVE MAXIMUM AMBIENT TEMPERATURE
TO-99 (J)	80°C	7.1mW/°C
8-Pin Hermetic DIP (Z)	75°C	6.7mW/°C
8-Pin Plastic DIP (P)	36°C	5.6mW/°C

2. Absolute maximum ratings apply to both packaged parts and DICE, unless otherwise noted.
3. For supply voltages less than ±22V, the absolute maximum input voltage is equal to the supply voltage.

ELECTRICAL CHARACTERISTICS at $V_S = \pm 15V$, $T_A = 25°C$, unless otherwise noted.

PARAMETER	SYMBOL	CONDITIONS	OP-77A MIN	OP-77A TYP	OP-77A MAX	OP-77B MIN	OP-77B TYP	OP-77B MAX	UNITS
Input Offset Voltage	V_{OS}		—	10	25	—	20	60	µV
Long-Term Input Offset Voltage Stability	ΔV_{OS}/Time	(Note 1)	—	0.2	—	—	0.2	—	µV/Mo
Input Offset Current	I_{OS}		—	0.1	1.5	—	0.1	2.8	nA
Input Bias Current	I_B		-0.2	1.2	2.0	-0.2	1.2	2.8	nA
Input Noise Voltage	e_{np-p}	0.1Hz to 10Hz (Note 2)	—	0.35	0.6	—	0.35	0.6	μV_{p-p}
Input Noise Voltage Density	e_n	f_O = 10Hz (Note 2)	—	10.3	18.0	—	10.3	18.0	nV/$\sqrt{Hz}$
		f_O = 100Hz (Note 2)	—	10.0	13.0	—	10.0	13.0	
		f_O = 1000Hz (Note 2)	—	9.6	11.0	—	9.6	11.0	
Input Noise Current	i_{np-p}	0.1Hz to 10Hz (Note 2)	—	14	30	—	14	30	pA_{p-p}
Input Noise Current Density	i_n	f_O = 10Hz (Note 2)	—	0.32	0.80	—	0.32	0.80	pA/$\sqrt{Hz}$
		f_O = 100Hz (Note 2)	—	0.14	0.23	—	0.14	0.23	
		f_O = 1000Hz (Note 2)	—	0.12	0.17	—	0.12	0.17	
Input Resistance — Differential-Mode	R_{IN}	(Note 3)	26	45	—	18.5	45	—	MΩ
Input Resistance — Common-Mode	R_{INCM}		—	200	—	—	200	—	GΩ
Input Voltage Range	IVR		±13	±14	—	±13	±14	—	V
Common-Mode Rejection Ratio	CMRR	V_{CM} = ±13V	—	0.1	1.0	—	0.1	1.6	µV/V
Power Supply Rejection Ratio	PSRR	V_S = ±3V to ±18V	—	1.0	3	—	1.0	3	µV/V
Large-Signal Voltage Gain	A_{VO}	$R_L \geq$ 2kΩ, V_O = ±10V	5000	12000	—	2000	8000	—	V/mV
Output Voltage Swing	V_O	$R_L \geq$ 10kΩ	±13.5	±14.0	—	±13.5	±14.0	—	V
		$R_L \geq$ 2kΩ	±12.5	±13.0	—	±12.5	±13.0	—	
		$R_L \geq$ 1kΩ	±12.0	±12.5	—	±12.0	±12.5	—	
Slew Rate	SR	$R_L \geq$ 2kΩ (Note 2)	0.1	0.3	—	0.1	0.3	—	V/µs
Closed-Loop Bandwidth	BW	A_{VCL} = +1 (Note 2)	0.4	0.6	—	0.4	0.6	—	MHz
Open-Loop Output Resistance	R_O	V_O = 0, I_O = 0	—	60	—	—	60	—	Ω
Power Consumption	P_d	V_S = ±15V, No Load	—	50	60	—	50	60	mW
		V_S = ±3V, No Load	—	3.5	4.5	—	3.5	4.5	
Offset Adjustment Range		R_P = 20kΩ	—	±3	—	—	±3	—	mV

NOTES:
1. Long-Term Input Offset Voltage Stability refers to the averaged trend line of V_{OS} vs. Time over extended periods after the first 30 days of operation. Excluding the initial hour of operation, changes in V_{OS} during the first 30 operating days are typically 2.5µV.

2. Sample tested.
3. Guaranteed by design.

+ 385-3771 Sherry

ELECTRICAL CHARACTERISTICS at $V_S = \pm 15V$, $-55°C \leq T_A \leq +125°C$, unless otherwise noted.

PARAMETER	SYMBOL	CONDITIONS	OP-77A MIN	TYP	MAX	OP-77B MIN	TYP	MAX	UNITS
Input Offset Voltage	V_{OS}		—	25	60	—	45	120	μV
Average Input Offset Voltage Drift	TCV_{OS}	(Note 1)	—	0.1	0.3	—	0.2	0.6	μV/°C
Input Offset Current	I_{OS}		—	0.1	2.2	—	0.1	4.5	nA
Average Input Offset Current Drift	TCI_{OS}	(Note 1)	—	0.5	25	—	0.5	50	pA/°C
Input Bias Current	I_B		-0.2	2.4	4	-0.2	2.4	6	nA
Average Input Bias Current Drift	TCI_B	(Note 1)	—	8	25	—	15	35	pA/°C
Input Voltage Range	IVR		±13	±13.5	—	±13	±13.5	—	V
Common-Mode Rejection Ratio	CMRR	$V_{CM} = \pm 13V$	—	0.1	10	—	0.1	3	μV/V
Power Supply Rejection Ratio	PSRR	$V_S = \pm 3V$ to $\pm 18V$	—	1	3	—	1	5	μV/V
Large-Signal Voltage Gain	A_{VO}	$R_L \geq 2k\Omega$, $V_O = \pm 10V$	2000	6000	—	1000	4000	—	V/mV
Output Voltage Swing	V_O	$R_L \geq 2k\Omega$	±12	±12.5	—	±12	±12.5	—	V
Power Consumption	P_d	$V_S = \pm 15V$, No Load	—	60	75	—	60	75	mW

NOTES:
1. Sample tested.
2. Guaranteed by design.

TYPICAL OFFSET VOLTAGE TEST CIRCUIT

TYPICAL LOW-FREQUENCY NOISE TEST CIRCUIT

OPTIONAL OFFSET NULLING CIRCUIT

BURN-IN CIRCUIT

OPERATIONAL AMPLIFIERS

PMI

ELECTRICAL CHARACTERISTICS at $V_S = \pm 15V$, $T_A = 25°C$, unless otherwise noted.

PARAMETER	SYMBOL	CONDITIONS	OP-77E MIN	OP-77E TYP	OP-77E MAX	OP-77F/G MIN	OP-77F/G TYP	OP-77F/G MAX	UNITS
Input Offset Voltage	V_{OS}		—	10	25	—	20	60	μV
Long-Term V_{OS} Stability	V_{OS}/Time	(Note 1)	—	0.3	—	—	0.4	—	μV/Mo
Input Offset Current	I_{OS}		—	0.1	1.5	—	0.1	2.8	nA
Input Bias Current	I_B		-0.2	1.2	2.0	-0.2	1.2	2.8	nA
Input Noise Voltage	e_{np-p}	0.1Hz to 10Hz (Note 2)	—	0.35	0.6	—	0.38	0.65	μV$_{p-p}$
Input Noise Voltage Density	e_n	$f_O = 10$Hz, $f_O = 100$Hz (Note 2), $f_O = 1000$Hz	—, —, —	10.3, 10.0, 9.8	18.0, 13.0, 11.0	—, —, —	10.5, 10.2, 9.8	20.0, 13.5, 11.5	nV/$\sqrt{\text{Hz}}$
Input Noise Current	i_{np-p}	0.1Hz to 10Hz (Note 2)	—	14	30	—	15	35	pA$_{p-p}$
Input Noise Current Density	i_n	$f_O = 10$Hz, $f_O = 100$Hz (Note 2), $f_O = 1000$Hz	—, —, —	0.32, 0.14, 0.12	0.60, 0.23, 0.17	—, —, —	0.35, 0.15, 0.13	0.90, 0.27, 0.18	pA/$\sqrt{\text{Hz}}$
Input Resistance — Differential-Mode	R_{IN}	(Note 3)	26	45	—	16.5	45	—	MΩ
Input Resistance — Common-Mode	R_{INCM}		—	200	—	—	200	—	GΩ
Input Voltage Range	IVR		±13	±14	—	±13	±14	—	V
Common-Mode Rejection Ratio	CMRR	$V_{CM} = \pm13V$	—	0.1	1.0	—	0.1	1.6	μV/V
Power Supply Rejection Ratio	PSRR	$V_S = \pm3V$ to $\pm18V$	—	1.0	3.0	—	1.0	3.0	μV/V
Large-Signal Voltage Gain	A_{VO}	$R_L \geq 2k\Omega$, $V_O = \pm10V$	5000	12000	—	2000	8000	—	V/mV
Output Voltage Swing	V_O	$R_L \geq 10k\Omega$, $R_L \geq 2k\Omega$, $R_L \geq 1k\Omega$	±13.8, ±12.5, ±12.0	±14.0, ±13.0, ±12.5	—, —, —	±13.8, ±12.5, ±12.0	±14.0, ±13.0, ±12.5	—, —, —	V
Slew Rate	SR	$R_L \geq 2k\Omega$ (Note 2)	0.1	0.3	—	0.1	0.3	—	V/μs
Closed-Loop Bandwidth	BW	$A_{VCL} = +1$ (Note 2)	0.4	0.6	—	0.4	0.6	—	MHz
Open-Loop Output Resistance	R_O	$V_O = 0$, $I_O = 0$	—	60	—	—	60	—	Ω
Power Consumption	P_d	$V_S = \pm15V$, No Load, $V_S = \pm3V$, No Load	—, —	50, 3.5	60, 4.5	—, —	50, 3.5	60, 4.5	mW
Offset Adjustment Range		$R_P = 20k\Omega$	—	±3	—	—	±3	—	mV

NOTES:
1. Long-Term Input Offset Voltage Stability refers to the averaged trend line of V_{OS} vs. Time over extended periods after the first 30 days of operation. Excluding the initial hour of operation, changes in V_{OS} during the first 30 operating days are typically 2.5μV.
2. Sample tested.
3. Guaranteed by design.

1/86, Rev. A

ELECTRICAL CHARACTERISTICS at $V_S = \pm15V$, $-25°C \leq T_A \leq +85°C$ for OP-77E/F, $0°C \leq T_A \leq +70°C$ for OP-77G, unless otherwise noted.

PARAMETER	SYMBOL	CONDITIONS	OP-77E MIN	OP-77E TYP	OP-77E MAX	OP-77F/G MIN	OP-77F/G TYP	OP-77F/G MAX	UNITS
Input Offset Voltage	V_{OS}		—	10	45	—	20	100	µV
Average Input Offset Voltage Drift	TCV_{OS}	(Note 1)	—	0.1	0.3	—	0.2	0.6	µV/°C
Input Offset Current	I_{OS}		—	0.1	2.2	—	0.1	4.5	nA
Average Input Offset Current Drift	TCI_{OS}	(Note 1)	—	0.5	40	—	0.5	85	pA/°C
Input Bias Current	I_B		-0.2	2.4	4.0	-0.2	2.4	6.0	nA
Average Input Bias Current Drift	TCI_B	(Note 1)	—	8	40	—	15	60	pA/°C
Input Voltage Range	IVR		±13.0	±13.5	—	±13.0	±13.5	—	V
Common-Mode Rejection Ratio	CMRR	$V_{CM} = \pm13V$	—	0.1	1.0	—	0.1	3.0	µV/V
Power Supply Rejection Ratio	PSRR	$V_S = \pm3V$ to $\pm18V$	—	1.0	3.0	—	1.0	5.0	µV/V
Large-Signal Voltage Gain	A_{VO}	$R_L \geq 2k\Omega$ $V_O = \pm10V$	2000	6000	—	1000	4000	—	V/mV
Output Voltage Swing	V_O	$R_L \geq 2k\Omega$	±12	±13.0	—	±12	±13.0	—	V
Power Consumption	P_d	$V_S = \pm15V$, No Load	—	60	75	—	60	75	mW

NOTES:
1. Sample tested.
2. Guaranteed by design.

OPEN-LOOP GAIN LINEARITY

Actual open-loop voltage can vary greatly at various output voltages. All automated testers use end-point testing and therefore only show the average gain. This causes errors in high closed-loop gain circuits. Since this is so difficult for manufacturers to test, you should make your own evaluation. This simple test circuit makes it easy. An ideal op amp would show a horizontal scope trace.

This is the output gain linearity trace for the new OP-77. The output trace is virtually horizontal at all points, assuring extremely high gain accuracy. The average open-loop gain is truly impressive — approximately 10,000,000.

OPERATIONAL AMPLIFIERS

5

DICE CHARACTERISTICS

1. BALANCE
2. INVERTING INPUT
3. NONINVERTING INPUT
4. V−
6. OUTPUT
7. V+
8. BALANCE

DIE SIZE 0.100 × 0.061 inch, 6100 sq. mils
(2.54 × 1.55 mm, 3.935 sq. mm)

For additional DICE information refer to
1986 Data Book, Section 2.

WAFER TEST LIMITS at $V_S = \pm 15V$, $T_A = 25°C$ for OP-77N/G devices.

PARAMETER	SYMBOL	CONDITIONS	OP-77N LIMIT	OP-77G LIMIT	UNITS
Input Offset Voltage	V_{OS}		40	75	μV MAX
Input Offset Current	I_{OS}		2.0	2.8	nA MAX
Input Bias Current	I_B		±2	±2.8	nA MAX
Input Resistance Differential-Mode	R_{IN}	(Note 1)	26	17	MΩ MIN
Input Voltage Range	IVR		±13	±13	V MIN
Common-Mode Rejection Ratio	CMRR	$V_{CM} = \pm 13V$	1	1.6	μV/V MAX
Power Supply Rejection Ratio	PSRR	$V_S = \pm 3V$ to $\pm 18V$	3	3	μV/V MAX
Output Voltage Swing	V_O	$R_L = 10k\Omega$ $R_L = 2k\Omega$ $R_L = 1k\Omega$	±13.5 ±12.5 ±12.0	±13.5 ±12.5 ±12.0	V MIN
Large-Signal Voltage Gain	A_{VO}	$R_L = 2k\Omega$ $V_O = \pm 10V$	2000	1000	V/mV MIN
Differential Input Voltage			±30	±30	V MAX
Power Consumption	P_d	$V_{OUT} = 0V$	60	60	mW MAX

NOTES:
1. Guaranteed by design.

Electrical tests are performed at wafer probe to the limits shown. Due to variations in assembly methods and normal yield loss, yield after packaging is not guaranteed for standard product dice. Consult factory to negotiate specifications based on dice lot qualification through sample lot assembly and testing.

TYPICAL ELECTRICAL CHARACTERISTICS at $V_S = \pm 15V$, $T_A = +25°C$, unless otherwise noted.

PARAMETER	SYMBOL	CONDITIONS	OP-77N TYPICAL	OP-77G TYPICAL	UNITS
Average Input Offset Voltage Drift	TCV_{OS}	$R_S = 50\Omega$	0.1	0.2	μV/°C
Nulled Input Offset Voltage Drift	TCV_{OSn}	$R_S = 50\Omega$, $R_P = 20k\Omega$	0.1	0.2	μV/°C
Average Input Offset Current Drift	TCI_{OS}		0.5	0.5	pA/°C
Slew Rate	SR	$R_L \geq 2k\Omega$	0.3	0.3	V/μs
Closed-Loop Bandwidth	BW	$A_{VCL} = +1$	0.6	0.6	MHz

MOTOROLA

MC34080/MC35080
thru
MC34085/MC35085

2

HIGH SLEW RATE, WIDE BANDWIDTH, JFET INPUT OPERATIONAL AMPLIFIERS

These devices are a new generation of high speed JFET input monolithic operational amplifiers. Innovative design concepts along with JFET technology provide wide gain bandwidth product and high slew rate. Well matched JFET input devices and advanced trim techniques ensure low input offset errors and bias currents. The all NPN output stage features large output voltage swing, no deadband crossover distortion, high capacitive drive capability, excellent phase and gain margins, low open-loop output impedance, and symmetrical source/sink ac frequency response.

This series of devices are available in standard or prime performance (A suffix) grades, fully compensated or decompensated ($A_{VCL} \geq 2$) and are specified over commercial or Military temperature ranges. They are pin compatible with existing industry standard operational amplifiers, and allow the designer to easily upgrade the performance of existing designs.

- Wide Gain Bandwidth: 8.0 MHz for Fully Compensated Devices
  16 MHz for Decompensated Devices

- High Slew Rate: 25 V/μs for Fully Compensated Devices
  50 V/μs for Decompensated Devices

- High Input Impedance: 10^{12} Ω

- Input Offset Voltage: 0.5 mV Maximum (Single Amplifier)

- Large Output Voltage Swing: − 14.7 V to + 14 V for
  $V_{CC}/V_{EE} = \pm 15$ V

- Low Open-Loop Output Impedance: 30 Ω @ 1.0 MHz

- Low THD Distortion: 0.01%

- Excellent Phase/Gain Margins: 55°/7.6 dB for Fully Compensated Devices

HIGH PERFORMANCE JFET INPUT OPERATIONAL AMPLIFIERS

P SUFFIX
PLASTIC PACKAGE
CASE 626-05

U SUFFIX
CERAMIC PACKAGE
CASE 693-02

D SUFFIX
PLASTIC PACKAGE
CASE 751-02
SO-8

PIN ASSIGNMENTS

Offset Null [1] — [8] NC
Invt Input [2] — [7] V_{CC}
Noninvt Input [3] — [6] Output
V_{EE} [4] — [5] Offset Null

Single. Top View

Output 1 [1] — [8] V_{CC}
Inputs 1 — [7] Output 2
V_{EE} [4] — Inputs 2

Dual. Top View

DW SUFFIX
PLASTIC PACKAGE
CASE 751G-01
SO-16L

Output 1 [1] — [16] Output 4
Outputs 1 [2]/[3] — [15]/[14] Inputs 4
V_{CC} [4] — [13] V_{EE}
Inputs 2 [5]/[6] — [12]/[11] Inputs 3
Output 2 [7] — [10] Output 3
NC [8] — [9] NC

ORDERING INFORMATION

Op Amp Function	Fully Compensated	$A_{VCL} \geq 2$ Decompensated	Temperature Range	Package
Single	MC35081U,AU	MC35080U,AU	−55 to +125°C	Ceramic DIP
	MC34081D,AD	MC34080D,AD	0 to +70°C	SO-8
	MC34081P,AP	MC34080P,AP	0 to +70°C	Plastic DIP
Dual	MC34082P,AP	MC34083P,AP	0 to +70°C	Plastic DIP
Quad	MC35084L,AL	MC35085L,AL	−55 to +125°C	Ceramic DIP
	MC34084DW	MC34085DW	0 to +70°C	SO-16L
	MC34084P,AP	MC34085P,AP	0 to +70°C	Plastic DIP

P SUFFIX
PLASTIC PACKAGE
CASE 646-06

L SUFFIX
CERAMIC PACKAGE
CASE 632-08

PIN ASSIGNMENTS

Output 1 [1] — [14] Output 4
Inputs 1 [2]/[3] — [13]/[12] Inputs 4
V_{CC} [4] — [11] V_{EE}
Inputs 2 [5]/[6] — [10]/[9] Inputs 3
Output 2 [7] — [8] Output 3

Quad. Top View

MOTOROLA LINEAR/INTERFACE DEVICES

2-283

MC34080, MC35080 Series

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage (from V_{CC} to V_{EE})	V_S	+44	Volts
Input Differential Voltage Range	V_{IDR}	Note 1	Volts
Input Voltage Range	V_{IR}	Note 1	Volts
Output Short-Circuit Duration (Note 2)	t_S	Indefinite	Seconds
Operating Ambient Temperature Range MC35XXX MC34XXX	T_A	 −55 to +125 0 to +70	°C
Operating Junction Temperature Ceramic Package Plastic Package	T_J	 +165 +125	°C
Storage Temperature Range Ceramic Package Plastic Package	T_{stg}	 −65 to +165 −55 to +125	°C

NOTES:
1. Either or both input voltages must not exceed the magnitude of V_{CC} or V_{EE}.
2. Power dissipation must be considered to ensure maximum junction temperature (T_J) is not exceeded.

EQUIVALENT CIRCUIT SCHEMATIC (EACH AMPLIFIER)

DC ELECTRICAL CHARACTERISTICS ($V_{CC} = +15$ V, $V_{EE} = -15$ V, $T_A = T_{low}$ to T_{high} [Note 3], unless otherwise noted)

Characteristic	Symbol	A Suffix			Non-Suffix			Unit
		Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage (Note 4)	V_{IO}							mV
Single								
$T_A = +25°C$		—	0.3	0.5	—	0.5	1.0	
$T_A = 0°C$ to $+70°C$ (MC34080, MC34081)		—	—	2.5	—	—	3.0	
$T_A = -55°C$ to $+125°C$ (MC35080, MC35081)		—	—	3.5	—	—	4.0	
Dual								
$T_A = +25°C$		—	0.6	1.0	—	1.0	3.0	
$T_A = 0°C$ to $+70°C$ (MC34082, MC34083)		—	—	3.0	—	—	5.0	
$T_A = -55°C$ to $+125°C$ (MC35082, MC35083)		—	—	4.0	—	—	6.0	
Quad								
$T_A = +25°C$		—	3.0	6.0	—	6.0	12	
$T_A = 0°C$ to $+70°C$ (MC34084, MC34085)		—	—	8.0	—	—	14	
$T_A = -55°C$ to $+125°C$ (MC35084, MC35085)		—	—	9.0	—	—	15	
Average Temperature Coefficient of Offset Voltage	$\Delta V_{IO}/\Delta T$	—	10	—	—	10	—	$\mu V/°C$
Input Bias Current ($V_{CM} = 0$ Note 5)	I_{IB}							nA
$T_A = +25°C$		—	0.06	0.2	—	0.06	0.2	
$T_A = 0°C$ to $+70°C$		—	—	4.0	—	—	4.0	
$T_A = -55°C$ to $+125°C$		—	—	50	—	—	50	
Input Offset Current ($V_{CM} = 0$ Note 5)	I_{IO}							nA
$T_A = +25°C$		—	0.02	0.1	—	0.02	0.1	
$T_A = 0°C$ to $+70°C$		—	—	2.0	—	—	2.0	
$T_A = -55°C$ to $+125°C$		—	—	25	—	—	25	
Large Signal Voltage Gain ($V_O = \pm10$ V, $R_L = 2.0$ k)	A_{VOL}							V/mV
$T_A = +25°C$		50	80	—	25	80	—	
$T_A = T_{low}$ to T_{high}		25	—	—	15	—	—	
Output Voltage Swing	V_{OH}							V
$R_L = 2.0$ k, $T_A = +25°C$		13.2	13.7	—	13.2	13.7	—	
$R_L = 10$ k, $T_A = +25°C$		13.4	13.9	—	13.4	13.9	—	
$R_L = 10$ k, $T_A = T_{low}$ to T_{high}		13.4	—	—	13.4	—	—	
$R_L = 2.0$ k, $T_A = +25°C$	V_{OL}	—	-14.1	-13.5	—	-14.1	-13.5	
$R_L = 10$ k, $T_A = +25°C$		—	-14.7	-14.1	—	-14.7	-14.1	
$R_L = 10$ k, $T_A = T_{low}$ to T_{high}		—	—	-14.0	—	—	-14.0	
Output Short-Circuit Current ($T_A = +25°C$)	I_{SC}							mA
Input Overdrive = 1.0 V, Output to Ground								
Source		20	31	—	20	31	—	
Sink		20	28	—	20	28	—	
Input Common Mode Voltage Range $T_A = +25°C$	V_{ICR}	$(V_{EE} + 4.0)$ to $(V_{CC} - 2.0)$			$(V_{EE} + 4.0)$ to $(V_{CC} - 2.0)$			V
Common Mode Rejection Ratio ($R_S \leq 10$ k, $T_A = +25°C$)	CMRR	75	90	—	70	90	—	dB
Power Supply Rejection Ratio ($R_S = 100 \Omega$, $T_A = 25°C$)	PSRR	75	86	—	70	86	—	dB
Power Supply Current	I_D							mA
Single								
$T_A = +25°C$		—	2.5	3.4	—	2.5	3.4	
$T_A = T_{low}$ to T_{high}		—	—	4.2	—	—	4.2	
Dual								
$T_A = +25°C$		—	4.9	6.0	—	4.9	6.0	
$T_A = T_{low}$ to T_{high}		—	—	7.5	—	—	7.5	
Quad								
$T_A = +25°C$		—	9.7	11	—	9.7	11	
$T_A = T_{low}$ to T_{high}		—	—	13	—	—	13	

NOTES (CONTINUED)
3. $T_{low} = -55°C$ for MC35080,A $T_{low} = 0°C$ for MC34080,A $T_{high} = +125°C$ for MC35080,A $T_{high} = +70°C$ for MC34080,A
       MC35081,A   MC34081,A   MC35081,A   MC34081,A
       MC35082,A   MC34082,A   MC35082,A   MC34082,A
       MC35083,A   MC34083,A   MC35083,A   MC34083,A
       MC35084,A   MC34084,A   MC35084,A   MC34084,A
       MC35085,A   MC34085,A   MC35085,A   MC34085,A
4. See application information for typical changes in input offset voltage due to solderability and temperature cycling.
5. Limits at $T_A = +25°C$ are guaranteed by high temperature (T_{high}) testing.

2

AC ELECTRICAL CHARACTERISTICS (V_{CC} = +15 V, V_{EE} = -15 V, T_A = +25°C unless otherwise noted)

Characteristic	Symbol	A Suffix			Non-Suffix			Unit
		Min	Typ	Max	Min	Typ	Max	
Slew Rate (V_{in} = -10 V to +10 V, R_L = 2.0 k, C_L = 100 pF)	SR							V/μs
Compensated A_V = -1.0		20	25	—	20	25	—	
A_V = -1.0		—	30	—	—	30	—	
Decompensated A_V = -2.0		40	50	—	40	50	—	
A_V = -1.0		—	50	—	—	50	—	
Settling Time (10 V Step, A_V = -1.0)	t_s							μs
To 0.10% (± ½ LSB of 9-Bits)		—	0.72	—	—	0.72	—	
To 0.01% (± ½ LSB of 12-Bits)		—	1.6	—	—	1.6	—	
Gain Bandwidth Product (f = 200 kHz)	GBW							MHz
Compensated		6.0	8.0	—	6.0	8.0	—	
Decompensated		12	16	—	12	16	—	
Power Bandwidth (R_L = 2.0 k, V_O = 20 V_{p-p}, THD = 5.0%)	BWp							kHz
Compensated A_V = -1.0		—	400	—	—	400	—	
Decompensated A_V = -1.0		—	800	—	—	800	—	
Phase Margin (Compensated)	ϕm							Degrees
R_L = 2.0 k		—	55	—	—	55	—	
R_L = 2.0 k, C_L = 100 pF		—	39	—	—	39	—	
Gain Margin (Compensated)	A_m							dB
R_L = 2.0 k		—	7.6	—	—	7.6	—	
R_L = 2.0 k, C_L = 100 pF		—	4.5	—	—	4.5	—	
Equivalent Input Noise Voltage R_S = 100 Ω, f = 1.0 kHz	e_n	—	30	—	—	30	—	nV/$\sqrt{Hz}$
Equivalent Input Noise Current (f = 1.0 kHz)	i_n	—	0.01	—	—	0.01	—	pA/$\sqrt{Hz}$
Input Capacitance	C_i	—	5.0	—	—	5.0	—	pF
Input Resistance	r_i	—	10^{12}	—	—	10^{12}	—	Ω
Total Harmonic Distortion A_V = +10, R_L = 2.0 k, 2.0 ≤ V_O ≤ 20 V_{p-p}, f = 10 kHz	THD	—	0.05	—	—	0.05	—	%
Channel Separation (f = 10 kHz)	—	—	120	—	—	120	—	dB
Open-Loop Output Impedance (f = 1.0 MHz)	z_o	—	35	—	—	35	—	Ω

TYPICAL PERFORMANCE CURVES

FIGURE 1 — INPUT COMMON MODE VOLTAGE RANGE versus TEMPERATURE

FIGURE 2 — INPUT BIAS CURRENT versus TEMPERATURE

2

FIGURE 3 — INPUT BIAS CURRENT versus
INPUT COMMON-MODE VOLTAGE

FIGURE 4 — OUTPUT VOLTAGE SWING
versus SUPPLY VOLTAGE

FIGURE 5 — OUTPUT SATURATION versus
LOAD CURRENT

FIGURE 6 — OUTPUT SATURATION versus
LOAD RESISTANCE TO GROUND

FIGURE 7 — OUTPUT SATURATION versus
LOAD RESISTANCE TO V_{CC}

FIGURE 8 — OUTPUT SHORT CIRCUIT CURRENT
versus TEMPERATURE

2

FIGURE 9 — OUTPUT IMPEDANCE versus FREQUENCY

FIGURE 10 — OUTPUT IMPEDANCE versus FREQUENCY

FIGURE 11 — OUTPUT VOLTAGE SWING versus FREQUENCY

FIGURE 12 — OUTPUT DISTORTION versus FREQUENCY

FIGURE 13 — OPEN-LOOP VOLTAGE GAIN versus TEMPERATURE

2

FIGURE 14 — OPEN-LOOP VOLTAGE GAIN AND PHASE versus FREQUENCY

FIGURE 15 — OPEN-LOOP VOLTAGE GAIN AND PHASE versus FREQUENCY

FIGURE 16 — OPEN-LOOP VOLTAGE GAIN AND PHASE versus FREQUENCY

FIGURE 17 — NORMALIZED GAIN BANDWIDTH PRODUCT versus TEMPERATURE

FIGURE 18 — PERCENT OVERSHOOT versus LOAD CAPACITANCE

FIGURE 19 — PHASE MARGIN versus LOAD CAPACITANCE

2

FIGURE 20 — GAIN MARGIN versus LOAD CAPACITANCE

FIGURE 21 — PHASE MARGIN versus TEMPERATURE

FIGURE 22 — GAIN MARGIN versus TEMPERATURE

FIGURE 23 — NORMALIZED SLEW RATE versus TEMPERATURE

2

MC34084 TRANSIENT RESPONSE
$A_V = +1.0$, $R_L = 2.0$ k, $V_{CC}/V_{EE} = \pm 15$ V, $T_A = 25°C$

FIGURE 24 — SMALL-SIGNAL

Vertical = 50 mV/Div
Horizontal 0.2 μs/Div
$C_L = 10$ pF

0 —

FIGURE 25 — LARGE-SIGNAL

Vertical = 5.0 V/Div
Horizontal = 0.5 μs/Div
$C_L = 100$ pF

0 —

MC34085 TRANSIENT RESPONSE
$A_V = +2.0$, $R_L = 2.0$ k, $V_{CC}/V_{EE} = \pm 15$ V, $T_A = 25°C$

FIGURE 26 — SMALL-SIGNAL

Vertical = 50 mV/Div
Horizontal = 0.2 μs/Div
$C_L = 10$ pF

0 —

FIGURE 27 — LARGE-SIGNAL

Vertical = 5.0 V/Div
Horizontal = 0.5 μs/Div
$C_L = 100$ pF

0 —

2

FIGURE 28 — COMMON-MODE REJECTION RATIO
versus FREQUENCY

FIGURE 29 — POWER SUPPLY REJECTION RATIO
versus FREQUENCY

FIGURE 30 — POWER SUPPLY REJECTION RATIO
versus TEMPERATURE

FIGURE 31 — NORMALIZED SUPPLY CURRENT
versus SUPPLY VOLTAGE

FIGURE 32 — CHANNEL SEPARATION versus FREQUENCY

FIGURE 33 — SPECTRAL NOISE DENSITY

2

APPLICATIONS INFORMATION

The bandwidth and slew rate of the MC34080 series is nearly double that of currently available general purpose JFET op-amps. This improvement in ac performance is due to the P-channel JFET differential input stage driving a compensated miller integration amplifier in conjunction with an all NPN output stage.

The all NPN output stage offers unique advantages over the more conventional NPN/PNP transistor Class AB output stage. With a 10 k load resistance, the op-amp can typically swing within 1.0 V of the positive rail (V_{CC}), and within 0.3 volts of the negative rail (V_{EE}), providing a 28.7 Vp–p swing from ±15 volt supplies. This large output swing becomes most noticeable at lower supply voltages. If the load resistance is referenced to V_{CC} instead of ground, the maximum possible output swing can be achieved for a given supply voltage. For light load currents, the load resistance will pull the output to V_{CC} during the positive swing and the NPN output transistor will pull the output very near V_{EE} during the negative swing. The load resistance value should be much less than that of the feedback resistance to maximize pull-up capability.

The all NPN transistor output stage is also inherently fast, contributing to the operational amplifier's high gain-bandwidth product and fast settling time. The associated high frequency output impedance is 50 ohms (typical) at 8.0 MHz. This allows driving capacitive loads from 0 to 300 pF without oscillations over the military temperature range, and over the full range of output swing. The 55° phase margin and 7.6 dB gain margin as well as the general gain and phase characteristics are virtually independent of the sink/source output swing conditions. The high frequency characteristics of the MC34080 series is especially useful for active filter applications.

The common mode input range is from 2.0 volts below the positive rail (V_{CC}) to 4.0 volts above the neg-

ative rail (V_{EE}). The amplifier remains active if the inputs are biased at the positive rail. This may be useful for some applications in that single supply operation is possible with a single negative supply. However, a degradation of offset voltage and voltage gain may result.

Phase reversal does not occur if either the inverting or noninverting input (or both) exceeds the positive common mode limit. If either input (or both) exceeds the negative common mode limit, the output will be in the high state. The input stage also allows a differential up to ±44 volts, provided the maximum input voltage range is not exceeded. The supply voltage operating range is from ±5.0 V to ±22 V.

For optimum frequency performance and stability careful component placement and printed circuit board layout should be exercised. For example, long unshielded input or output leads may result in unwanted input-output coupling. In order to reduce the input capacitance, resistors connected to the input pins should be physically close to these pins. This not only minimizes the input pole for optimum frequency response, but also minimizes extraneous "pickup" at this node.

Supply decoupling with adequate capacitance close to the supply pin is also important, particularly over temperature, since many types of decoupling capacitors exhibit large impedance changes over temperature.

Primarily due to the JFET inputs of the op amp, the input offset voltage may change due to temperature cycling and board soldering. After 20 temperature cycles (–55°C to 165°C), the typical standard deviation for input offset voltage is 558 μV and 473 μV in the plastic and ceramic packages respectively. With respect to board soldering (260°C, 10 seconds) the typical standard deviation for input offset voltage is 525 μV and 227 μV in the plastic and ceramic package respectively. Socketed plastic or ceramic packaged devices should be used over a minimal temperature range for optimum input offset voltage performance.

FIGURE 34 — OFFSET NULLING CIRCUIT

Appendix G -   **Power Supply Board**

Appendix G - **PCB layouts**

OUTBB

OUTDB

OUTFIP

OUTALT

OUTAS

OUTSY

U4

U5

F2

F3

F1

F1

F5

F6

REL2

SW1

F7

REL1

U1

U2

U7

M1

M2

M3

Appendix G -   **Parts Data Sheets**

Customer Application Technical Summary
VI-100/200 DC to DC Converters

THERMAL CONSIDERATIONS

Vicor manufactures standard component power converters, that are the analog to very efficient three terminal regulators. Cooling must be provided by external means. The factor that must be controlled by the system designer is the thermal resistance of the baseplate to free air, which in turn determines the baseplate operating temperature. The thermal resistance of the baseplate to air can be controlled by maintaining the ambient temperature, moving air over the baseplate, adding a heatsink to the module, or mounting the module to a cold plate such as the system chassis.

The heat that is dissipated by a Vicor converter is related to the output power of the module and the efficiency of the module. Vicor converters are among the most efficient available today. They are also the smallest. The effective dissipation per square inch of surface area can be very high, relative to traditional types of supplies, but remains very low compared to power transistors and complex logic circuits. For additional information, consult Vicor's application note: Cooling High Density DC/DC Converters.

The following variables should be considered in the design of an effective cooling system for Vicor modules:

- Efficiency of the module (refer to data sheet)
- Thermal resistance, baseplate to environment
- Maximum ambient temperature
- Amount of air flow, as it relates to thermal resistance

Figure 1. Thermal Resistance, Baseplate to Free Air

Air Flow	VI-100/VI-200 (No Heat Sink)
Free Air	5.00°C/W
100 LFM	3.30°C/W
200 LFM	2.00°C/W
300 LFM	1.38°C/W
400 LFM	1.26°C/W
500 LFM	1.15°C/W
600 LFM	1.07°C/W
700 LFM	0.99°C/W
800 LFM	0.91°C/W

VI-100/200 ADJUSTMENT PROCEDURE

Definition: Adjustment range of the output voltage, as a percent of nominal.

It should be noted that several specifications are a function of nominal output voltage settings, such as efficiency, ripple and input voltage range. In general as the output voltage is trimmed down, efficiency goes down, ripple as a percent of V_{OUT} goes up (although actual peak to peak level remains essentially constant) and lastly, input voltage range widens since input dropout (loss of regulation) moves down. As the units are trimmed up the reverse of the above effects occur.

All Vicor converters have a fixed current limit. As the output voltage is trimmed down the current limit set point remains constant. Therefore, in terms of output power, if the unit is trimmed down 20%, the available output power drops by 20%. Do not exceed maximum power rating when unit is trimmed up in voltage.

VI-200 converters have a very wide range of adjustability that can prove beneficial to the user. The output of a VI-200 can be reduced to zero volts with secondary side only circuitry. VI-200's can be used in power amplifier applications requiring fast programmability. VI-200 converters exhibit a rise time of approximately 10mS. In many instances this feature will provide a solution for those systems requiring "odd" output voltages. VI-200 converters are switch-mode power supplies, therefore a load is required to program the output voltage to zero volts. Consult catalog for additional information.

Although VI-100 converters may be trimmed down in excess of 20%, the specified adjustment range is − 20%. VI-100's are not characterized for operation below this range.

Both series of converters have a practical limit of + 10% on trim range, due to a fixed over-voltage set point. Although they can be adjusted up higher, the output may run into OVP. In practice there should be 0.5 Volts minimum between V_{OUT} and OVP, ensuring that OVP will not be activated during transient events. This is very important on VI-200 converters, since the OVP is of a latching type.

Please refer to Figure 2, for the correct adjustment procedure for VI-100 and VI-200 series converters.

Figure 2. Adjustment Procedure

VI-100 Internal Values-Output Voltage					
Volts	5V	12V	15V	24V	48V
R2	1 0	1 9	2 5	8 6	18 2
R3	1 0	1 0	1 0	1 0	10 0
	1 0	0 48	0 48	1 0	1 0

K OHMS

VI-100 Trim Values					
Range	+ 10% −20%	+ 10% −20%	+ 10% −20%	+ 10% −20%	+9% −20%
R6	0 150	1 2	2 2	6 7	25 0
R7	1 0	1 0	1 0	1 0	10 0
R8	0 56	0 33	0 39	0 68	0

K OHMS

VI-200, Mega and Master Modules Internal Values-Output Voltage					
Volts	5V	12V	15V	24V	48V
R2	20	76	100	172	364
R3	20	20	20	20	20

K OHMS

VI-200, Mega and Master Modules Trim Values					
Range	+ 10% −100%	+ 10% −100%	+ 10% −100%	+ 10% −100%	+ 10% −100%
R6	9 2	35	46	79	167
R7	10	10	10	10	10

K OHMS

R1 and R4 vary as a function of input voltage. output voltage and output power

VI-100

VI-200

RECOMMENDED TRIM VALUES FOR + 10%. − 100% TRIM RANGE:

$$R6 = \frac{1.1 \times V_{nom} - 2.75}{3} \times 10K$$

REMOTE SENSE LINES

Vicor converters provide two pins (+ S, − S) that allow the output error amplifier to sense output voltage so that output voltage may be accurately defined at one point. This point may be "local" to the module, or it may be "remote" at the actual point of load. If the sense pins are left unconnected, the output voltage will be above nominal and load regulation will be poor. Although sense lines may be of any length, the converter will only compensate for a finite drop in the output lines. Consult data sheet for remote sense compensation specifications. Long sense lines should be twisted or otherwise shielded to minimize noise pickup. If the module is affected by noise pickup, capacitors should be added from each sense pin to its respective output to close the feedback loop locally (AC).

CONNECTION OF BOOSTER MODULES

Power boosters are used in applications requiring higher output current than an individual driver module is capable of providing. VI-200 driver modules may be used as boosters, by trimming up the output voltage 10%. This essentially allows the unit to be controlled by another driver module. Current sharing between driver and booster converters occurs naturally regardless of the number of boosters. Any number of boosters may be added to a driver module. Boosters **must** be of the same family (VI-100 or VI-200) and of the same input voltage, output voltage and output power.

Connect the converters as follows:

NOTES:

. NC = Do not connect, internal circuitry may be present.

. R1, R2 installed if trim desired. See Adjustment Procedure.

. ⬤ are installed only if drivers used in respective positions.

SAFETY AGENCY CONSIDERATIONS (VI-200 Series)

In order to meet the requirements of UL 478, CSA 22.2 and TUV (IEC 380), certain precautions must be excercised.

1. **Input Voltage:** Do not exceed input voltage rating of converter.
2. **Baseplate:** If the baseplate of the converter is accessible to the operator of the equipment, ground the baseplate to chassis ground.
3. **Temperature:** Under normal operating conditions the temperature of the baseplate must be 85 C or less at the middle mounting slot of the converter. Temperature must be verified at maximum system load and maximum, system specified, ambient temperature.

4. **Trimming:** Do not trim output voltage higher than 110% of nominal rated output voltage. Do not exceed rated output power when unit is trimmed up. If converter is trimmed down, maximum output current is constant, therefore total available output power will be less than rated output power. Observe the same precautions when using output sense.

5. **Overtemperature Shutdown:** If the internal temperature of the module exceeds 90-105°C the module will shut down. The module must be cooled down and input voltage recycled to re-start the converter.

6. **External Protection:** Required. Each converter must be fused separately with: Bussman Mfg. Co., PC-Tron 250V, Max. 3A.

ADDITIONAL FILTERING - IF DESIRED

Vicor's zero-current-switching converters **do not** generally require additional filtering. In applications where particularly low levels of noise are specified, some of the components shown below may be employed to advantage. Consult Vicor's applications engineering.

NOTES:

1. Specific values vary by model.

2. C3 is Aluminum Electrolytic (Lossy).

3. If L3, C8 corner frequency is <65 KHz, use R1, C7.

4. R1, C7 (if used) corner frequency should be <1 kHz

5. C4, 1000pf ceramic with short leads.

GENERAL APPLICATION INFORMATION

Label: There is no need to remove the label on Vicor products. The thermal drop across the label is very low. VI-200 series converters may be ordered without a label if desired. If the converter is mounted to a chassis or heatsink, apply a 5 mil coating of thermal compound to the label surface, or use GRAFOIL thermally conductive sheeting. Consult the Vicor Product Catalog for ordering information.

Reverse Input Voltage: Two methods exist to protect against reverse inputs. A diode may be added in series with the input or in the alternative, fuse the individual converter with a fuse rated at no more the 2X maximum input current at low line, and add a shunt diode across the input.

External Voltage Applied To Outputs: External voltages up to 125% of nominal output may be applied when unit is either unpowered or disabled.

Common Mode Noise: The small amount of common mode noise current flowing between primary and secondary may in some instances cause the module to fail to deliver full power or to make audible noise. In this event bypass the baseplate of the converter by installing a 1000pf ceramic capacitor, with short leads, between (–) output and the base and (–) input and the base.

Measuring Output Ripple: Output differential ripple is the AC component present on the output that is not common to an output or its return. The most accurate way to measure this parameter is to power the converter from a battery. A battery is used to virtually eliminate common-mode noise that is often confused with differential output noise. If powering the converter from a battery is not feasible, a reasonably accurate measurement can still be made by keeping the scope ground lead as short as possible. The ground lead of a scope has inductance that is not present in the signal lead. This differential inductance converts the common-mode noise into differential noise as viewed by the scope. Vicor converters have a half sine power pulse that is filtered through an LC. The module contains no discontinuities in secondary side voltage waveforms that create high dv/dt's causing output "spikes", virtually eliminating output "spikes".

VICOR
CORPORATION

23 Frontage Rd. Andover, MA 01810

TEL: (617) 470-2900
FAX: (617) 475-6715
TWX: 910-380-5144

Effective July 16, 1988
our area code (617) will
change to (508).

Customer Application Technical Summary
VI-100/200 DC to DC Converters

THERMAL CONSIDERATIONS

Vicor manufactures standard component power converters, that are the analog to very efficient three terminal regulators. Cooling must be provided by external means. The factor that must be controlled by the system designer is the thermal resistance of the baseplate to free air, which in turn determines the baseplate operating temperature. The thermal resistance of the baseplate to air can be controlled by maintaining the ambient temperature, moving air over the baseplate, adding a heatsink to the module, or mounting the module to a cold plate such as the system chassis.

The heat that is dissipated by a Vicor converter is related to the output power of the module and the efficiency of the module. Vicor converters are among the most efficient available today. They are also the smallest. The effective dissipation per square inch of surface area can be very high, relative to traditional types of supplies, but remains very low compared to power transistors and complex logic circuits. For additional information, consult Vicor's application note: Cooling High Density DC/DC Converters.

The following variables should be considered in the design of an effective cooling system for Vicor modules:

- Efficiency of the module (refer to data sheet)
- Thermal resistance, baseplate to environment
- Maximum ambient temperature
- Amount of air flow, as it relates to thermal resistance

Figure 1. Thermal Resistance, Baseplate to Free Air

Air Flow	VI-100/VI-200 (No Heat Sink)
Free Air	5.00°C/W
100 LFM	3.30°C/W
200 LFM	2.00°C/W
300 LFM	1.38°C/W
400 LFM	1.26°C/W
500 LFM	1.15°C/W
600 LFM	1.07°C/W
700 LFM	0.99°C/W
800 LFM	0.91°C/W

VI-100/200 ADJUSTMENT PROCEDURE

Definition: Adjustment range of the output voltage, as a percent of nominal.

It should be noted that several specifications are a function of nominal output voltage settings, such as efficiency, ripple and input voltage range. In general as the output voltage is trimmed down, efficiency goes down, ripple as a percent of V_{OUT} goes up (although actual peak to peak level remains essentially constant) and lastly, input voltage range widens since input dropout (loss of regulation) moves down. As the units are trimmed up the reverse of the above effects occur.

All Vicor converters have a fixed current limit. As the output voltage is trimmed down the current limit set point remains constant. Therefore, in terms of output power, if the unit is trimmed down 20%, the available output power drops by 20%. Do not exceed maximum power rating when unit is trimmed up in voltage.

VI-200 converters have a very wide range of adjustability that can prove beneficial to the user. The output of a VI-200 can be reduced to zero volts with secondary side only circuitry. VI-200's can be used in power amplifier applications requiring fast programmability. VI-200 converters exhibit a rise time of approximately 10mS. In many instances this feature will provide a solution for those systems requiring "odd" output voltages. VI-200 converters are switch-mode power supplies, therefore a load is required to program the output voltage to zero volts. Consult catalog for additional information.

Although VI-100 converters may be trimmed down in excess of 20%, the specified adjustment range is − 20%. VI-100's are not characterized for operation below this range.

Both series of converters have a practical limit of + 10% on trim range, due to a fixed over-voltage set point. Although they can be adjusted up higher, the output may run into OVP. In practice there should be 0.5 Volts minimum between V_{OUT} and OVP, ensuring that OVP will not be activated during transient events. This is very important on VI-200 converters, since the OVP is of a latching type.

Please refer to Figure 2, for the correct adjustment procedure for VI-100 and VI-200 series converters.

Figure 2. Adjustment Procedure

VI-100

VI-100 Internal Values-Output Voltage					
Volts	5V	12V	15V	24V	48V
R2	1 0	1 9	2 5	8 6	18 2
R3	1 0	1 0	1 0	1 0	10 0
	1 0	0 48	0 48	1 0	1 0

K OHMS

VI-100 Trim Values					
Range	+ 10% − 20%	+ 10% − 20%	+ 10% − 20%	+ 10% − 20%	+ 9% − 20%
R6	0 150	1 2	2 2	6 7	25 0
R7	1 0	1 0	1 0	1 0	10 0
R8	0 56	0 33	0 39	0 68	0

K OHMS

VI-200, Mega and Master Modules Internal Values-Output Voltage					
Volts	5V	12V	15V	24V	48V
R2	20	76	100	172	364
R3	20	20	20	20	20

K OHMS

VI-200, Mega and Master Modules Trim Values					
Range	+ 10% − 100%	+ 10% − 100%	+ 10% − 100%	+ 10% − 100%	+ 10% − 100%
R6	9 2	35	46	79	167
	10	10	10	10	10

K OHMS

R1 and R4 vary as a function of input voltage, output voltage and output power.

RECOMMENDED TRIM VALUES FOR + 10%, − 100% TRIM RANGE:

$$R6 = \frac{1.1 \times V_{nom} − 2.75}{3} \times 10K$$

REMOTE SENSE LINES

Vicor converters provide two pins (+ S, – S) that allow the output error amplifier to sense output voltage so that output voltage may be accurately defined at one point. This point may be "local" to the module, or it may be "remote" at the actual point of load. If the sense pins are left unconnected, the output voltage will be above nominal and load regulation will be poor. Although sense lines may be of any length, the converter will only compensate for a finite drop in the output lines. Consult data sheet for remote sense compensation specifications. Long sense lines should be twisted or otherwise shielded to minimize noise pickup. If the module is affected by noise pickup, capacitors should be added from each sense pin to its respective output to close the feedback loop locally (AC).

CONNECTION OF BOOSTER MODULES

Power boosters are used in applications requiring higher output current than an individual driver module is capable of providing. VI-200 driver modules may be used as boosters, by trimming up the output voltage 10%. This essentially allows the unit to be controlled by another driver module. Current sharing between driver and booster converters occurs naturally regardless of the number of boosters. Any number of boosters may be added to a driver module. Boosters **must** be of the same family (VI-100 or VI-200) and of the same input voltage, output voltage and output power.

Connect the converters as follows:

NOTES:

1. NC = Do not connect, internal circuitry may be present.

2. R1, R2 installed if trim desired. See Adjustment Procedure.

3. ● *are installed only if drivers used in respective positions.*

SAFETY AGENCY CONSIDERATIONS (VI-200 Series)

In order to meet the requirements of UL 478, CSA 22.2 and TUV (IEC 380), certain precautions must be excercised.

1. **Input Voltage:** Do not exceed input voltage rating of converter.

2. **Baseplate:** If the baseplate of the converter is accessible to the operator of the equipment, ground the baseplate to chassis ground.

3. **Temperature:** Under normal operating conditions the temperature of the baseplate must be 85 C or less at the middle mounting slot of the converter. Temperature must be verified at maximum system load and maximum, system specified, ambient temperature.
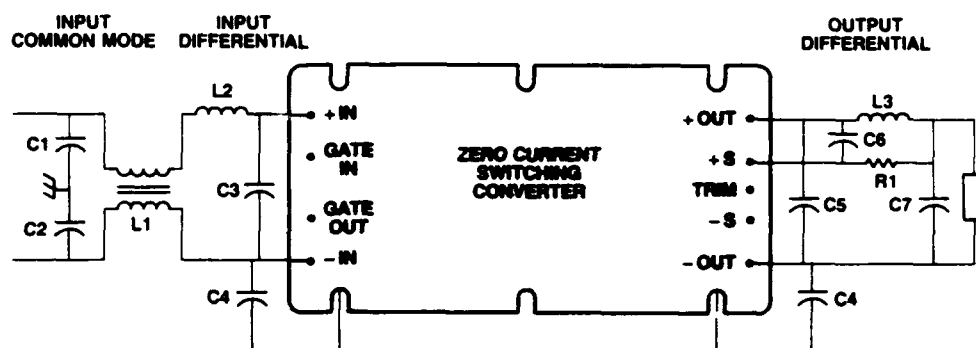
4. **Trimming:** Do not trim output voltage higher than 110% of nominal rated output voltage. Do not exceed rated output power when unit is trimmed up. If converter is trimmed down, maximum output current is constant, therefore total available output power will be less than rated output power. Observe the same precautions when using output sense.

5. **Overtemperature Shutdown:** If the internal temperature of the module exceeds 90-105°C the module will shut down. The module must be cooled down and input voltage recycled to re-start the converter.

6. **External Protection:** Required. Each converter must be fused separately with: Bussman Mfg. Co., PC-Tron 250V, Max. 3A.

ADDITIONAL FILTERING - IF DESIRED

Vicor's zero-current-switching converters **do not** generally require additional filtering. In applications where particularly low levels of noise are specified, some of the components shown below may be employed to advantage. Consult Vicor's applications engineering.

NOTES:

1. Specific values vary by model.

2. C3 is Aluminum Electrolytic (Lossy).

3. If L3, C8 corner frequency is <65 KHz, use R1, C7.

4. R1, C7 (if used) corner frequency should be <1 kHz

5. C4, 1000pf ceramic with short leads.

GENERAL APPLICATION INFORMATION

Label: There is no need to remove the label on Vicor products. The thermal drop across the label is very low. VI-200 series converters may be ordered without a label if desired. If the converter is mounted to a chassis or heatsink, apply a 5 mil coating of thermal compound to the label surface, or use GRAFOIL thermally conductive sheeting. Consult the Vicor Product Catalog for ordering information.

Reverse Input Voltage: Two methods exist to protect against reverse inputs. A diode may be added in series with the input or in the alternative, fuse the individual converter with a fuse rated at no more the 2X maximum input current at low line, and add a shunt diode across the input.

External Voltage Applied To Outputs: External voltages up to 125% of nominal output may be applied when unit is either unpowered or disabled.

Common Mode Noise: The small amount of common mode noise current flowing between primary and secondary may in some instances cause the module to fail to deliver full power or to make audible noise. In this event bypass the baseplate of the converter by installing a 1000pf ceramic capacitor, with short leads, between (−) output and the base and (−) input and the base.

Measuring Output Ripple: Output differential ripple is the AC component present on the output that is not common to an output or its return. The most accurate way to measure this parameter is to power the converter from a battery. A battery is used to virtually eliminate common-mode noise that is often confused with differential output noise. If powering the converter from a battery is not feasible, a reasonably accurate measurement can still be made by keeping the scope ground lead as short as possible. The ground lead of a scope has inductance that is not present in the signal lead. This differential inductance converts the common-mode noise into differential noise as viewed by the scope. Vicor converters have a half sine power pulse that is filtered through an LC. The module contains no discontinuities in secondary side voltage waveforms that create high dv/dt's causing output "spikes", virtually eliminating output "spikes".

Appendix H -   **Display Board**

Appendix H - **PCB Schematics**

Appendix H - **PCB layouts**

Display Board RJS 2.0 PCB

Wed, Aug 9, 1989   5:43 PM

Solder Side   Scale150%

Display Board RJS 2.0 PCB

Wed, Aug 9, 1989 5:43 PM

Component Xray   Scale150%

Display Board 2.0 Layout
Wed, Aug 9, 1989 5:08 PM
Component Xray   Scale150%

Appendix H -   **Parts Data Sheets**

ACCULEX
A METRABYTE COMPANY

4 1/2 Digit Multi-range, High Resolution, Voltage Input DPM with Enhanced Contrast LCD Display

FEATURES

- Multiple Ranges (User Configurable)
- Large (10 mm), 4 1/2 Digit Enhanced Contrast LCD Display
- Very Low Power Consumption (17.5 milliWatts)
- Choice of Decimal Point Placement
- Extreme Accuracy (0.06%, ±1 Digit)
- Common Mode Rejection of 86 db (min)
- Very Low Cost
- World's Smallest Size

APPLICATIONS

- Automotive/Marine/Avionics/Aerospace
- Field/Mobile Instrumentation
- Pharmaceutical Manufacturing
- Electronic Test
- Biochemical/Biomedical Research & Product Development
- Petrochemical Process Management
- Energy/Environmental Management
- Critical Accuracy Voltage Measurements

FUNCTIONAL DESCRIPTION

The DP-2000 is a very compact, 4 1/2 digit, low power (17.5 milliWatts), LCD Digital Panel Meter covering a broad range of critical voltage measurement applications for the industrial sector, the Original Equipment Instrument Manufacturer, and other areas where high accuracy and precise resolution over extended periods are crucial. The combination of VLSI components and ACCULEX's well known dedication to advanced Surface Mount Technology (SMT) results in the most reliable, lowest cost, highest resolution LCD Digital Panel Meter available from any source.

The DP-2000 is a multi-range voltage input DPM with user configurable full scale input ranges (±200 mV or ±2 Vdc) and choice of decimal point placement. Using a dual slope integrating A/D converter, differential input, and common mode rejection to 86 db (minimum), the DP-2000 is a rock solid, state-of-the-art DPM designed to be used in virtually any environment. Its extremely large (10 mm numeral height) enhanced contrast 4 1/2 digit LCD display means easy readability in dim light as well as direct sunlight conditions. Drawing only 3.5 mA @ +5 Vdc, the DP-2000 is perfect for field/mobile instruments, marine, aerospace/avionics, and automotive applications as well as for general purpose laboratory use or harsh, factory floor environments. The DP-2000 features a "LOW BATTERY" indicator (low voltage) as well as automatic polarity and over/under range indication. A "HOLD DISPLAY" pin is also standard and may be wired to a momentary contact (pushbutton) switch for temporarily "freezing" the display. The DP-2000 shares the same overall dimensions and panel cut-out size as the rest of the ACCULEX micro-size LCD DPM family.

Some of the more common application areas for the DP-2000 are petrochemical processing, pharmaceutical manufacturing, biochemical/biomedical research and product development, plastics manufacturing and processing, avionics, marine instrumentation, automotive instrumentation, food processing, energy management, environmental chambers, electronic test and burn-in chambers, power supply monitoring, power transmissions, medical instrumentation and much more.

ACCULEX is proud to offer the DP-2000 as the world's smallest, most accurate, and lowest cost 4 1/2 digit LCD Digital Panel Meter in the world. And ... like all ACCULEX products, the DP-2000 is available from stock in quantity or singles.

BLOCK DIAGRAM

DP-2000 PINOUT

Noise Rejection Switch

50 Hz

60 Hz

SPAN ADJ

12 — NC
HOLD DISPLAY
RANGE SELECT
DP COMMON
DP 4
DP 3
DP 2
DP 1
SIG IN (–)
SIG IN (+)
GND
1 — + 5 Vdc

RANGE SELECT: ±200 mV (Pin 10 OPEN) low range
±2 Vdc (Pin 10 to +5Vdc) high range

HOLD: Pin 11 to +5 Vdc

SPAN ADJ: May be adjusted at zero Vdc

NOISE REJECTION SWITCH: May be used to decrease 50/60 Hz line noise.

SPECIFICATIONS

Analog I/O

Input Configuration	Bipolar, Fully Differential
Full Scale Input	DP-2000: ±200 mV
	±2 Vdc
	DP-2002: ±2 Vdc
	±20 Vdc
	DP-2020: ±20 Vdc
	±200 Vdc
Sample Interval	2 readings per second
Accuracy	0.06% of FS ±1 Digit
A/D Converter Type	Dual Slope Integration
Input Bias Current	50 pA (typical)
Common Mode Voltage	±1 Vdc (max)
Common Mode Rejection	86 db (min)
Input Polarity	Bipolar, automatic changeover
Input Impedance	DP-2000; 1000 M Ohms
	DP-2002; 1 Meg Ohms
	DP-2020; 10 Meg Ohms
Input Voltage	±10 Vdc (max) (DP-2000)
	±100V (DP-2002)
	±350V (DP-2020)
Temperature Coef.	±50 ppm/Deg C (typ) (DP-2000)
	±75 ppm/Deg C (all others)

Display

Display Type	Enhanced Contrast LCD
Number of Digits	4 1/2 (19999, max)
Display Size	10 mm
Over Range Indication	1
Under Range Indication	–1
Display Polarity	Positive not displayed (" – " for negative)
Display Hold	Pin 11 tied to +5 Vdc (pin 1)
Display Update	Pin 11 Open
Decimal Point	4 Positions (Selectable)

Electrical / Environmental

Supply Voltage	+5 Vdc (±7%) @ 3.5 mA
Other	Low Battery Indication (Low Voltage)
Storage Temp	–10 to 60 Deg C
Operating Temp	0 to 50 Deg C
Weight	30 grams
Dimensions	2.33"(W)×1.14"(H)×0.83"(D)
	(59×29×21 mm)

OUTLINE DRAWING

-1.8.8.8.8

29
26
7.4
UNIT IN MM

59

3
18.5
7 3
2.54
56

ORDERING GUIDE

MODEL NO.	RANGE	PRICE (SINGLES)
DP-2000	±200 mV ±2 Vdc	$95.00
DP-2002	±2 Vdc ±20 Vdc	$95.00
DP-2020	±20 Vdc ±200 Vdc	$95.00
C-12 (Connector)		$ 4.00

QUANTITY PRICING (PER UNIT)

MODEL	5+	10+	25+	50+	100+
DP-2000	$90	$85	$79	$73	Call
C-12	$4.00	$3.50	$3.25	$3.00	

DIMENSIONS, PANEL CUT-OUT, AND MOUNTING SUGGESTIONS

DP-176:
The DP-176 may be mounted with or without the optional Bezel. Use the
following diagram and dimensions for accurate mounting.

DIMENSIONS (MM/IN.)

6.1
.24

.76
.03

$\frac{60}{2.36}$

$\frac{2.5}{.098}$ D(4)

$\frac{30}{1.18}$

$\frac{38.1}{1.50}$

BEZEL OPTION

$\frac{2.5}{.010}$ R max (4)

PANEL CUTOUT

$\frac{32.5}{1.28}$

$\frac{62.5}{2.46}$

$\frac{34.3}{1.35}$

$\frac{64.5}{2.54}$

6.1
.24

9
.35

DP-650, 660, 670, and DP-2000:
All 4 of these micro-size DPMs have the same external dimensions (L &
W). Only the depth is different. The suggested mounting procedure is
quite straightforward since they have an integrated bezel with "Rabbit-
ear" type mount. The DP-660, 670, and 2000 have a depth of 18.5 mm
whereas the DP-650's depth is 8.5 mm. The following panel cut-out di-
mensions should be used for accurate mounting.

$2.21 \begin{matrix} +0.010 \\ -0.000 \end{matrix}$

$\left(56 \begin{matrix} +0.05 \\ -0.00 \end{matrix}\right)$

PANEL
CUT-OUT

$1.02 \begin{matrix} +0.010 \\ -0.000 \end{matrix}$

$\left(26 \begin{matrix} +0.05 \\ -0.00 \end{matrix}\right)$

$\frac{1.1}{(29)}$

$\frac{0.29}{(7.4)}$

$\frac{1.0}{(26)}$

$\frac{0.25}{(6.35)}$ DIA.

$\frac{0.35}{(8.9)}$

$\frac{0.12}{(3)}$

$\frac{0.26}{(6.6)}$

Appendix I -   Mizar 8605 Analog Input Board Manual

Mizar Inc.

MZ 8605 Analog Input Module

User's Manual
Board Revision(s) C

Publication No. 7101-00033-0001

If you have any technical questions about this product, call Mizar Inc. at
(612) 224-8941 and ask for the "Technical Support Center".

Fourth Edition
Copyright 1986 by Mizar Inc.

MIZAR, INC.
VME Bus Modules & Systems
STD Bus Modules & Systems

Thank You

Thank you for being a Mizar customer. Mizar would like you to join our Engineering Design Team by inviting your suggestions on its products and documentation. Please complete and return this self addressed stamped Customer Report Form.

Product No./Rev. No. _____ Manual Rev. _____

COMMENTS: _____

_____

Current VME or STD Application: _____

Name_____ Title_____

Company_____ Phone___(___)_____

Address_____

City_____

State_____ Zip_____

Mail Drop_____

If you have a question or suggestion that requires immediate attention, please contact Mizar's Engineering Design Team at our Technical Support Center: (612) 224-8941.

UNPACKING

This board contains components that are susceptible to static discharge, and should be handled with appropriate caution.

Upon receipt of this product, visually inspect the board for missing, broken or damaged components and for physical damage to the printed circuit board or connectors.

This product was shipped in perfect condition. Any damage to the product is the responsibility of the shipping carrier and should be reported to the carriers agent immediately.

WARRANTY INFORMATION

Mizar warrants that the articles furnished hereunder are free from defects in material and workmanship and will perform as specified by Mizar for one year from date of shipment. This warranty is in lieu of any other warranty, expressed or implied. In no event will Mizar be liable for special or consequential damages as a result of any alleged breach of this warranty provision. The liability of Mizar hereunder shall be limited to repair or replacement, at manufacture's discretion, of any defective unit. Equipment or parts which have been subject to abuse, misuse, accident, alteration, neglect, unauthorized repair or installation are not covered by warranty. Mizar shall have the right of final determination as to the existence and cause of defect.

DISCLAIMER

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Mizar reserves the right to make changes to any products herein to improve reliability, function or design. Mizar does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

DATA SHEET ACKNOWLEDGEMENTS

The HS574 A/D converter data sheets are reprinted with the permission of the Hybrid Systems Corporation.

The AD524 Precision Amplifier data sheets are reprinted with the permission of Analog Devices Inc.

TABLE OF CONTENTS

1.1 BOARD SPECIFICATIONS

Data Transfer Mode: A16:D16

Interrupt Levels: Any one of 1 to 7

Operating Conditions: Temperature: 0 to 70° C
                        Humidity: 90%

Power Consumption: 1.2 A at 5VDC typical

Physical Size: 160mm by 100mm Single Height Eurocard

1.2 VME8605 FUNCTIONAL DESCRIPTION

The 8605 analog input board provides the user with 16 single ended or 8 double ended analog inputs(or optionally 32 single or 16 double ended inputs.) A general description of the board is best illustrated by figure 1.

Figure 1.
Block Diagram

The analog multiplexor (1) selects the appropriate analog input channel the board is to convert next. The selected signal is then applied single or double ended across the instrumentation amplifier (2) which can be set for gains of 1, 10, 100, 1000 or any arbitrary resistor programmed gain. After being amplified, the input signal is then sampled (and held during the conversion process) (3), then converted to the digital value by the successive approximation analog to digital converter (4). When the conversion is complete the interrupt logic (5), (if enabled), informs the CPU of a completed conversion.

CHAPTER 2

CONFIGURATION GUIDE

2.1 INTRODUCTION

The VME8605 can be configured for a variety of applications using the on-board jumper blocks labeled K1 - K10. These jumpers are used to specify the base address, interrupt request levels, amplifier gain, single and differential input, and analog-to-digital control signals. The purpose of this chapter is to describe the placement of these jumper blocks on the board and to explain how to set them for individual user applications.

In the following sections, the pins in the diagrams and on the board are numbered such that pin 1 is in the upper-left corner and the numbers increase left-to-right and top to bottom when the component side is up and the P1 connector is closest to the user. This convention may or may not agree with the schematic.

2.2 COMPONENT PLACEMENT and JUMPER BLOCK PLACEMENT

Figure 2 shows the location of the various components used on VME8605. Figure 3 shows the location of the jumper blocks on the board.

Figure 2.
Component
Placement

that this block (shown in figure 3) is configured in binary and that a jumper in place represents a zero in that respective bit, and a missing jumper represents a one in that respective bit. Figure 3 also shows an example of K10 and K8 configured for an IRQ level of 4.

K8 & K10: IRQ CONFIGURATION

```
           —— IRQ INPUT ——        K8: INT. ACK.
   PIN 1  □ □ □ □ □ □ □           B2  □ □
     K10  □ □ □ □ □ □ □           B1  □ □
 IRQ LEVEL 1 2 3 4 5 6 7          B0  □ □


      EXAMPLE: INTERRUPT REQUEST LEVEL 4
           —— IRQ INPUT ——        K8: INT. ACK.
   PIN 1  □ □ □ ▯ □ □ □           B2  □ □
     K10  □ □ □ ▯ □ □ □  (4)      B1  ▢—▢
 IRQ LEVEL 1 2 3 4 5 6 7          B0  ▢—▢
```

2.5 K2: Input Amplifier Gain

The instrumentation amplifier gain is set using jumper block K2 (and an external resistor is desired.) Figure 4 shows the pinout of K2. For proper jumper configuration see the Analog Devices AD524 data sheet attached. Note that an optional resistor can be placed in position R6 to set other gains. (R6 is connected from RG1 to RG2.) For a gain of one, remove all jumpers from K2.

K2: INPUT AMPLIFIER GAIN

```
              PIN 1
        RG1   □ □   RG2
        G10   □ □   RG2
       G100   □ □   RG2
      G1000   □ □   RG2
              PIN 7
```

2.6 K1 & K7: Single/Differential Input Selection

Two jumper blocks K1 and K7 determine the input mode of the board. Below is shown the proper jumper connections of K1 and K7 for single and differential input operation.

K1 &K7: SINGLE/DIFFERENTIAL INPUT

```
         K1          K7                          K1          K7
  PIN 1 ▢—▢          □              PIN 1      ▢—▢          ▯
  SINGLE ▮ ▮                     DIFFERENTIAL  ▢—▢          ▯
  ENDED  ▮ ▮          □                        □ □
  PIN 5                              PIN 5
```

CHAPTER 3

PROGRAMMING GUIDE

3.1 INTRODUCTION

This chapter will cover a variety of topics relating to the proper programming and operation of the VME8605 board. Included will be discussions on register addresses, conversion operations, calibration and other topics.

3.2 REGISTERS

The 8605 has 6 registers beginning at the base address as follows:

Address	Name	R/W	Function	Length (bits)
0	Converstrt	W	Start a conversion	16 (all ignored)
2	Chanselect	W	Select a channel and start a conversion	16 (upper 11 ignored)
2	AnalogRD	R	Analog to digital value	16 (lower 12 valid)
4	Intlevel	W	Interrupt vector	16 (upper 8 ignored)
6	Intenable	W	Enables interrupts	16 (all ignored)
8	Intdisable	W	Disables interrupts	16 (all ignored)

Note that all registers are 16 bits long although some bits are ignored. For proper operation, all writes and reads to the board must be 16 bits long, i.e. WORD operations. If other than word operations are performed, the board will not respond.

3.3.1 Conversion Operation

Note that to start a conversion either the converstrt register can be written to or the channel select register can be written to with the selected input channel. Once a conversion has been started, do not write to the chanselect or converstrt registers--this will re-start the conversion before it has a chance to complete.

The analogRD register is organized as shown in Figure x below.

| CIP | MSB | MSB | MSB | MSB | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | LSB |

The convert in process (CIP) bit is set if a conversion is taking place and clears when done. Polling this bit until it clears is one way of testing for a complete conversion. Note that the most significant bit of the conversion is brought out to bits 11 through 14, and that all of these bits may be optionally inverted by jumper block K6.

3.4 CALIBRATION

Calibration of the 8605 consists of nulling the instrumentation amplifier, the sample and hold circuits, and adjusting the offset and gain of the analog to digital converter. To begin, jumper all blocks as desired except blocks K1 and k2. Refer to figure 2 for resistor locations.

3.4.1 Instrumentation Amplifier Nulling Procedure

1. Set the gain of the instrumentation amplifier to one--remove all jumpers from K2 and remove R3 if it is in place.
2. Jumper pin 2 to pins 4 and 6 of K1.
3. Adjust the output offset null (R4) until pin 9 of U6 is 0.000 V.
4. Set the gain of the instrumentation amp to 1000--place a jumper across pins 7 and 8 of K2.
5. Adjust the input offset null (R2) until pin 9 of U6 is 0.000 V.
6. Repeat steps 1 through 5 until a satisfactory null point is reached

3.4.2 Sample and Hold Nulling

Because the sample and hold network is in the hold mode except when a conversion is started, a small software routine is required to cause the network to continuously sample. This is done by a loop that continuously starts a conversion. With this loop running and pin 9 of U6 near zero, adjust R7 until the output of the sample and hold network (pin 1 or 4 of K3)is the same as pin 9 of U6.

3.4.3 Analog to Digital Chip Calibration

The best description of analog to digital calibration is presented in the attached Analog Devices AD574 data sheet. To perform the calibration a small software routine is needed to continuously perform conversions and read the results (This can be the same routine used in the sample and hold nulling section.) Run this routine and examine the converted value for use in the procedure outlined in the data sheet.

3.4.4 Resistor table

R3: Optional instrumentation amplifier gain resistor.
R5: Input offset null for the instrumentation amplifier.
R6: Output offset null for the instrumentation amplifier.
R7: Sample and hold offset null.
R10: Analog to digital unipolar offset.
R11: Analog to digital bipolar offset.
R12: Analog to digital gain.

3.8 IMPORTANT NOTE FOR 8605 Revisions A, B, and C

Due to an irregularity in the timing of the Hybrid Systems analog to digital converter, it is possible that the Convert in Process bit (CIP bit) may indicate valid data up to 100nS before the data in the analogRD register is actually valid. This would cause false data to be read if the read operation occurred during this interval.

To avoid this situation, it is suggested that the analogRD register be read again after the CIP bit is detected low. This delay will guarantee valid data for the second read.

Note that in interrupt mode, the built in delay of the IACK cycle (triggered off of CIP) is sufficient to provide valid data the first time the analogRD register is read.

3.9 Programming Examples

The following program is a straight forward example of starting the conversion process and checking for the CIP bit an appropriate amount of time until the process is finished. If an error should occur, the program will halt execution.

*** Define Constants

```
iospace    = $ffff0000              * address of board
```

*** Program Begins

```
          lea iospace,a1           * load reg. a1 with short I/O address
          clr.l d0                 * initialize registers
          clr.l d1                 * ...
          move.w #$00,2(a1)        * start conversion on channel 0
          move.l #$200,d1          * set counter
readlop:  move.w 2(a1),d3          * read AnalogRD
          bpl convek               * if CIP bit = 0, conversion is done
                                   * digital value is in lower 12 bits
          dbf d1,readlop           * decrement and branch 200 times waiting
readerr:  stop #$2700              * conversion not done, abort operation
convok:   ....                     * continue with program
```

3.9.1 Interrupt Program Example

This next example shows one of the ways to set up and use interrupts. Please note that the address for 'vect6' is specific to Mizar's VME8900 debug monitor and may need to be different for other applications.

CHAPTER 4

THEORY OF OPERATION

4.1 INTRODUCTION

This chapter provides the user with a brief overview of the operation of the VME8605 board, giving insight into how the board performs some of its various functions. It is provided for those who desire a more detailed explanation of the processes involved and is not required for the understanding or programming of the board's functions.

It is assumed that the reader has some knowledge of the VMEbus Specifications and that he has a general knowledge of the interaction between electrical components. The user should refer to the schematic and the PAL equations during the following discussion.

4.2 ADDRESSING and DATA OPERATIONS

The VME8605 is designed to respond to 16 bit addresses in the VMEbus short I/O addressing range. This addressing range is specified by address modifier codes $2D or $29, one of which is outputted by the CPU depending on whether it is in the user or supervisor state.

Jumper block K9 specifies the high byte of the board address determined by the user and has its signals fed into U25, an octal comparator. If a match is found between K9 and address lines A8 - A15, pin 19 of U25 will go low. This signal, called /HAM is fed into PAL U26 where it used in conjunction with the address modifier codes to determine if the 8605 is being selected. If it is U26 generates /AMATCH which feeds into PAL U21 and is used along with address line A1 - A3 to determine which chip or register is desired. These address lines come from the VMEbus via U27, an octal bus driver. Note that U26 also acts as buffer for the read/write line from the VMEbus.

Data lines between the bus and the board are controlled by U28 and U29, octal bus tranceivers. The direction of data flow is controlled by a signal called /BUSRD which comes from PAL U22. If BUSRD is low, data goes onto the bus from the board, if BUSRD is high the board receives data from the bus. These chips are activated whenever data goes to/form the bus.

U19 and U23 are octal drivers that buffer the data lines between U10, the HS574 chip, and the bus data buffers. These chips are enabled by the signal /AREAD from PAL U21 which indicates that the HS574 is to send/receive some data. U24 is used as a register to hold the interrupt vector value. U18 is the data buffer for the channel select value enabled by /CHSEL from PAL U21.

Each of the above chips are activated by their respective signals which are generated from a combination of /AMATCH, several control signals from the VMEbus and a particular combination on address lines A1 - A3.

APPENDIX

APPENDIX A

HS 574 DATA SHEETS

Hybrid Systems

HS 574

Complete 12-Bit, 25μSec A/D Converter with μP Interface

FEATURES

- Complete 12-Bit A/D Converter with Reference, Clock and Three-state Outputs
- Full 8- or 16-Bit Microprocessor Bus Interface
- 150nSec Bus Access Time
- Guaranteed Linearity Over Temperature
- No Missing Codes Over Temperature
- Fast Conversion — 25μSec
- Precision Reference for Long-Term Stability and Low Gain T.C.
- Hermetic 28-Pin Metal or Low Cost Epoxy DIP
- Low Power: 600mW

DESCRIPTION

The HS 574 is a complete 12-bit successive-approximation analog-to-digital converter with three-state output buffers for direct interface to 8- or 16-bit microprocessor buses. The HS 574 is implemented with advanced bipolar and CMOS LSI chips resulting in maximum performance at lowest cost. The SAR, 12-bit decoded D/A, control logic, switches and buffers are fabricated using CMOS processing for lowest power. A unique comparator, reference and required amplifiers are fabricated using linear bipolar processes for maximum speed and reduced offset and drift over temperature.

Incorporating a unique precision comparator design, the HS 574 offers several advantages over more conventional circuits. Advantages include lower input impedance variation from device to device, faster conversion, lower initial offset, and lower parametric drift over temperature. A proprietary decoded 12-bit D/A provides increased accuracy, lower drift and reduced output noise over the A/D operating range. Precision low TCR laser trimmed resistors are used in the

converter for setting critical performance parameters including gain, offset, input ranges, and accuracy.

HS 574 is offered in a hermetically sealed package for use over a wide temperature range and for MIL-STD-883 requirements. The lower cost proprietary commercial package is offered for applications not requiring the wider temperature exposure.

The HS 574 is available in 6 product grades. The HS 574J, K and L are specified over a temperature range of 0°C to +70°C while the HS 574S, T and U are specified over the MIL temperature range –55°C to +125°C. All "B" versions of the HS 574 are fully screened to MIL-STD-883B and are processed in accordance with Method 5008.1. All units that are not specified as "B" (883B processing) are rigorously tested including full power burn in at +85°C. Hybrid Systems guarantees Acceptable Quality Level (AQL) of 0.4% for all commercial models which means that there are no rejects in a sample lot of 100 pieces. That's more than twice as tough as it has to be — even for military applications

FUNCTIONAL DIAGRAM

PACKAGE OUTLINE

Dimensions shown in inches and (mm)

CASE A

CASE B

*0.030 (0.76) ±0.010 (0.25) STANDOFFS,
SUPPLIED AT MANUFACTURERS OPTION.
** INDICATES PIN 1 (METAL PACKAGE)

PIN ASSIGNMENTS

PIN	FUNCTION	PIN	FUNCTION
1	V_{LOGIC}	28	STS (STATUS)
2	$12/\bar{8}$	27	DB_{11}(MSB)
3	$\overline{CS}$	26	DB_{10}
4	A_0	25	DB_9
5	$R/\bar{C}$	24	DB_8
6	CE	23	DB_7
7	V_{CC}	22	DB_6
8	REF OUT	21	DB_5
9	ANA GND(AC)	20	DB_4
10	REF IN	19	DB_3
11	V_{EE}	18	DB_2
12	BIP OFF	17	DB_1
13	$10V_{IN}$	16	DB_0(LSB)
14	$20V_{IN}$	15	DIGITAL GND

ABSOLUTE MAXIMUM RATINGS

V_{CC} to Digital Common	0 to +16 5V
V_{EE} to Digital Common	0 to -16 5V
V_{LOGIC} to Digital Common	0 to +7V
Analog Common to Digital Common	± 1V
Control Inputs (CE. $\overline{CS}$. A_0. $12/\bar{8}$. $R/\bar{C}$) to Digital Common	-0 5V to V_{LOGIC} +0 5V
Analog Inputs (REF IN. BIP OFF. $10V_{IN}$) to Analog Common	± 16 5V
$20V_{IN}$ to Analog Common	± 24V
REF OUT	Indefinite short to common Momentary short to V_{CC}
Power Dissipation	750mW
Lead Temperature. Soldering	300°C. 10Sec

CONTROL FUNCTIONS

The HS 574 contains all control functions necessary to provide for complete microprocessor interface and also. stand alone operation including continuous conversions. All control functions are defined in Table 1 and Table 2.

Function	Definition	Function
CE	Chip Enable	1 Typically used as clock synchronization with μP. 2. Must be high (1) for a conversion to start. 3. Must be high (1) to read data on the output. 4 ⌐ transition may be used to initiate conversion
$\overline{CS}$	Chip Select	1. Typically the address pin when used with μP 2. Must be low (0) for a conversion to start or read data at the output. 3. ⌐ transition may be used to initiate conversion.
$R/\bar{C}$	Read/Convert	1. ⌐ initiate conversion 2. ⌐ initiate read
A_0	Address	1. Selects conversion mode 12 Bits if low (0). 8 Bits if high (1). 2. In read mode A_0 selects the output format. If low (0) then 8 MSB's (high and middle byte) or if high (1) then only low byte and trailing zeros.
$12/\bar{8}$	Output Format	1. May be hard wired. 2. Normal 12 Bit format if high (1). 3. 8-Bit format as set by A_0 if low (0).

Table 1. Defining the Control Functions

CONTROL INPUTS					HS 574 OPERATION
CE	$\overline{CS}$	$R/\bar{C}$	$12/\bar{8}$	A_0	
0	X	X	X	X	No Operation
X	1	X	X	X	No Operation
1	0	⌐	X	0	Initiates 12-Bit Conversion
1	0	⌐	X	1	Initiates 8-Bit Conversion
⌐	0	0	X	0	Initiates 12-Bit Conversion
⌐	0	0	X	1	Initiates 8-Bit Conversion
1	⌐	0	X	0	Initiates 12-Bit Conversion
1	⌐	0	X	1	Initiates 8-Bit Conversion
1	0	⌐	Pin 1	X	Enables 12-Bit Parallel Output
1	0	⌐	Pin 15	0	Enables 8 MSB's
1	0	⌐	Pin 15	1	Enables 4 LSB's and 4 Trailing Zeros

NOTES. 1. 1 indicates logic HIGH.
2. 0 indicates logic LOW.
3. X indicates don't care.
4. ⌐ indicates operation commences on low to high transition.
5. MSB → XXXX XXXX XXXX ← LSB
         High   Middle Low
         Byte   Byte   Byte

Table 2. HS 574 Truth Table

CONTINUOUS CONVERSION

Requirements for self triggered-continuous conversions are popular applications for an analog to digital converter, see Fig. 3

Figure 3. Continuous Conversion
Top: DATA BUS 12 Bits or greater
Bottom: DATA BUS 8 Bits

$\overline{CS}$ and A_0 are tied low (0) while $12/\overline{8}$ is tied high (1) to select the converter and enable a 12-Bit conversion. Note A_0 is 'don't care' in the truth table.

CE is connected to a 1kΩ and 0.1µF integrator as shown, this ensures an initial conversion on power up. CE will see a rising edge which will initiate a conversion ($\overline{CS} = 0$, $R/\overline{C} = 0$). The RC network will then integrate the initial 1 at the output of the first inverter causing a delay in the $R/\overline{C}$ command. After the first conversion, continuous conversions are caused by delaying the STATUS (STS) into $R/\overline{C}$. After the conversion is complete the output data lines come out of tri-state approximately 70nS after STS goes low (DONE). Data will remain valid (from previous conversion) 100nS after the new $R/\overline{C}$ 乁 command which allows for the positive edge 丆 triggered data to be loaded into the external buffer (75LS374 or equivalent).

Using the R-C network as shown, 1.5µS is allowed between conversions. Shorter times can be used but a longer time will cause long rise and fall of the $R/\overline{C}$ line and the clock input to the buffer. The setup time for the latch shown is 20nS and the hold time is 0nS

The user may access the octal latches asynchronously by means of the OUTPUT ENABLE (CONTROL OUTPUT) line. The data will always be valid, for a 12-bit conversion. Using this method, data will always be current and the STATUS bit need not be tested for valid data.

USING THE A₀ LINE

The state of the A_0 line at the start of a conversion places the HS 574 in either a full 12-bit conversion or in an 8-bit 'short cycle' mode. During a READ at the end of a conversion the A_0 line is used to the format of the data as follows:

1. Prior to Conversion

	MODE
$A_0 = 1$	Short cycle 8-bit conversion
$A_0 = 0$	Full 12-bit conversion

2. After Conversion (READ)

$A_0 = 1$	Data = Low Byte (LSB) followed by zeros
$A_0 = 0$	Data = High Byte (MSB's) followed by middle and low byte

In a µP application the A_0 line can be considered a pair of $\overline{WR}$ locations as follows:

1. Prior to Conversion (WRITE)

	MODE
$\overline{WR} = 0$ in low address ($A_0 = 0$)	Full 12-bit conversion
$\overline{WR} = 0$ in high address ($A_0 = 1$)	Short cycle 8-bit conversion

2. After Conversion (READ)

$\overline{WR} = 1$ in either address ($A_0 = X$)	Full 12-bit word with $12/\overline{8} = 1$
$\overline{WR} = 1$ in high address ($A_0 = 1$)	LSB's & zeros when $12/\overline{8} = 0$
$\overline{WR} = 1$ in low address ($A_0 = 0$)	8 MSB's only when $12/\overline{8} = 0$

INTERFACING THE HS 574 WITH 8-BIT MICROPROCESSORS

The HS 574 which has 12-bit data can be used directly with popular 8-bit microprocessors. The data however, must be multiplexed by setting the output mode select $12/\overline{8}$ pin to GND.

In the first case, a 6800 (or 6502) is used. See Figure 4.

Note 1. Decoding may be needed for a large system.

Figure 4. Interfacing the HS 574 and a 6800 µP

The STATUS (STS) is tied directly to $\overline{IREQ}$ which is the interrupt line. When STS goes to 0 (at the end of a conversion) the 6800 may either service the interrupt or be timed for 30µS (since this $\overline{IREQ}$ is software maskable) the time required for a conversion.

Figure 5 shows the 8080A µP as interfaced with the HS 574. In this case, a 8228 controller is shown with gates to generate needed signals.

Figure 5. Interfacing the HS 574 and 8080A µP

See
Note 2

HS 574

Pin	Name
2	12/8̄
3	C̄S̄
4	A₀
5	R/C̄
6	CE
10	REF IN
8	REF OUT
12	BIP OFF
13	10Vᵢₙ
14	20Vᵢₙ
9	ANA COM

Pin	Name
28	STS
24-27	HIGH BYTE
20-23	MIDDLE BYTE
16-19	LOW BYTE
1	+5V
7	+15V
11	-15V
15	DIG COM

2. To increase adjustment range:
   a) Change R3 to 33kΩ, and R2 to 300Ω.
   b) Add series resistor 100Ω to ±5V input and 200Ω to
      ±10V input.

*Figure 8b. Unipolar Input Connections
with Trim Adjustment*

ZERO ADJUSTMENT PROCEDURE

1. For unipolar ranges:
   a) Set input voltage precisely to +½LSB.
   b) Adjust zero control until converter is switching from
      000000000000 to 000000000001.

2. For bipolar ranges:
   a) Set input voltage precisely to ½LSB above — F S.
   b) Adjust zero control until converter is switching from
      000000000000 to 000000000001

GAIN ADJUSTMENT PROCEDURE

1. Set input voltage precisely to ½LSB less than 'all bits on' value.
   Note that this is 1½LSB less than nominal full scale.
2. Adjust gain control until converter is switching from
   111111111110 to 111111111111.

Table 4 summarizes the zero and gain adjustment procedure, and
shows the proper input test voltages used in calibrating the
HS 574

Input Voltage Range	Adjust-ment	Input Voltage	Adjust input to point where converter is just on the verge of switching between the two codes shown.[1]
0 to +20V	ZERO	2.44mV	000000000000 000000000001
0 to +20V	GAIN	19.9927V	111111111110 111111111111
0 to +10V	ZERO	1.22mV	000000000000 000000000001
0 to +10V	GAIN	9.9963V	111111111110 111111111111
±5V	ZERO	−4.9988V	000000000000 000000000001
±5V	GAIN	4.9933V	111111111110 111111111111
±10V	ZERO	−9.9976V	000000000000 000000000001
±10V	GAIN	9.9927V	111111111110 111111111111

[1]Codes shown are natural binary for unipolar input ranges and off-
set binary for bipolar ranges.

Table 4. Calibration Data

POWER SUPPLY CONSIDERATION

Power supplies used for the HS 574 should be selected for low
noise operation. In particular they should be free of high frequency
noise. Unstable output codes may result with noisy power sources. It
is important to remember that 2.44mV is 1LSB for a 10 volt input.

Decoupling capacitors are recommended on all power supply pins
located as close to the converter as possible. Suitable decoupling
capacitors are 10µF tantalum type in parallel with 0.1µF disc
ceramic type.

GROUNDING CONSIDERATIONS

The analog common at pin 9 is the ground reference point for the
internal reference and is thus the high quality ground for the
HS 574; it should be connected directly to the analog reference
point of the system. In order to achieve all of the high accuracy per-
formance available from the HS 574 in an environment of high
digital noise content, it is recommended that the analog and digital
commons be connected together at the package. In some situa-
tions, the digital common at pin 15 can be connected to the most
convenient ground reference point; analog power return is pre-
ferred. If digital common contains high frequency noise beyond
200mV, this noise may feed through the converter, so that some
caution will be required.

It is also important in the layout, to carefully consider the placement
of digital lines. It is recommended that digital lines not be run direct-
ly under the 574. For optimum system performance, if space
permits, a ground plane is advised under the 574. This should be
connected to a digital ground. Finally, in packaging the assembled
574, the designer should also try to minimize any capacitive
coupling that might occur at the top to the device.

APPENDIX B

AD524 DATA SHEETS

ANALOG DEVICES

Precision Instrumentation Amplifier

AD524

ADVANCE TECHNICAL DATA

FEATURES
Low Nonlinearity: 0.005% (G = 1)
High CMRR: 130dB (G = 1000)
Low Offset Voltage: 50µV
Low Offset Voltage Drift: 0.5µV/°C
Gain Bandwidth Product: 25MHz
Pin Programmable Gains of 1, 10, 100, 1000
Complete Input Protection, Power On – Power Off
No External Components Required
Internally Compensated

AD524 FUNCTIONAL BLOCK DIAGRAM

PRODUCT DESCRIPTION

The AD524 is a precision monolithic instrumentation amplifier designed for data acquisition applications requiring high accuracy under worst-case operating conditions. An outstanding combination of high linearity, high common mode rejection, low offset voltage drift, and low noise makes the AD524 suitable for use in many data acquisition systems.

The AD524 has an output offset voltage drift of less than 20µV/°C, input offset voltage drift of less than 0.5µV/°C, CMR above 90dB at unity gain (120dB at G = 1000) and maximum nonlinearity of 0.005% at G = 1. In addition to the outstanding dc specifications the AD524 also has a 25MHz gain bandwidth product (G = 100). To make it suitable for high speed data acquisition systems the AD524 has an output slew rate of 5V/µs and settles in 15µs up to a gain of 100.

As a complete amplifier the AD524 does not require any external components for fixed gains of 1, 10, 100 and 1,000. For other gain settings between 1 and 1000 only a single resistor is required. The AD524 input is fully protected for both power on and power off fault conditions.

The AD524 IC instrumentation amplifier is available in four different versions of accuracy and operating temperature range. The economical "J" grade, the low drift "K" grade and lower drift, higher linearity "L" grade are specified from 0 to +70°C. The "S" grade guarantees performance to specification over the full MIL-temperature range: –55°C to +125°C and is available screened to MIL-STD-883, Class B.

PRODUCT HIGHLIGHTS

1. The AD524 has low guaranteed offset voltage, offset voltage drift and low noise for precision high gain applications.

2. The AD524 is functionally complete with pin programmable gains of 1, 10, 100 and 1000.

3. Input and output offset nulling terminals are provided for very high precision applications and to minimize offset voltage changes in gain ranging applications.

4. The AD524 is fully input protected for both power on and power off fault conditions.

5. The AD524 offers superior dynamic performance with a gain bandwidth product of 25MHz, full peak response of 75kHz and a settling time of 15µs to 0.01% of a 10V step (G = 100).

Figure 1. Pin Configuration

Figure 2. Offset Null Circuits

Figure 3. Gain vs. Frequency

Figure 4. CMRR vs. Frequency RTI, Zero to 1k Source Imbalance

Figure 5. Slew Rate vs. Gain

Figure 6. PSRR vs. Frequency

SPECIFICATIONS (typical @ $V_S = \pm 15V$, $R_L = 2k\Omega$ and $T_A = +25°C$ unless otherwise specified)

Model	AD524J	AD524K	AD524L	AD524S
GAIN				
Gain Equation				
External Resistor Gain				
Programming)	$\frac{40,000}{R_G} - 1 \pm 20\%$	*	*	*
Gain Range	1 to 1000	*	*	*
Pin Programmable				
Gain Error, Max				
G = 1	±0.05%	±0.02%	±0.02%	*
G = 10	±0.25%	±0.1%	±0.05%	*
G = 100	±0.5%	±0.25%	±0.1%	*
G = 1000	±2%	±2%	±1%	*
Nonlinearity, max				
G = 1	±0.01%	*	±0.005%	*
G = 10	±0.01%	*	±0.005%	*
G = 100	±0.01%	*	*	*
G = 1000	±0.01%	*	*	*
Gain vs. Temperature				
G = 1	5ppm/°C	*	*	*
G = 10	5ppm/°C	*	*	*
G = 100	10ppm/°C	*	*	*
G = 1000	25ppm/°C	*	*	*
OUTPUT RATING	±10V @ 5mA	*	*	*
DYNAMIC RESPONSE				
Small Signal – 3dB				
G = 1	1MHz	*	*	*
G = 10	400kHz	*	*	*
G = 100	150kHz	*	*	*
G = 1000	25kHz	*	*	*
Slew Rate				
G = 1	2.5V/µs	*	*	*
G = 10–1000	5.0V/µs	*	*	*
Settling Time to 0.01%				
G = 1 to 10	15µs	*	*	*
G = 100	15µs	*	*	*
G = 1000	75µs	*	*	*
VOLTAGE OFFSET (Max to Noise)				
Input Offset Voltage, max	250µV	100µV	50µV	**
vs. Temperature, max	2µV/°C	1µV/°C	0.5µV/°C	2µV/°C
Output Offset Voltage, max	1mV	500µV	250µV	**
vs. Temperature, max	10µV/°C	50µV/°C	20µV/°C	25µV/°C
Offset Referred to the				
Input vs. Supply				
G = 1	75dB	80dB	80dB	**
G = 10	95dB	95dB	100dB	**
G = 100	105dB	105dB	105dB	**
G = 1000	105dB	110dB	115dB	**
INPUT CURRENT				
Input Bias Current, max	±50nA	±25nA	±15nA	*
vs. Temperature	±100pA/°C	*	*	*
Input Offset Current, max	±35nA	±15nA	±10nA	*
vs. Temperature	±100pA/°C	*	*	*
INPUT				
Input Impedance				
Differential	$10^9\Omega$	*	*	*
Common Mode	$10^9\Omega$	*	*	*
Input Voltage Range				
Max Differ. Input Linear	±10V	*	*	*
Max Common Mode Linear	±10V	*	*	*
Common Mode Rejection Ratio				
dc to 60Hz with 1kΩ Source Imbalance, min				
G = 1	70	75	80	*
G = 10	90	95	100	*
G = 100	110	115	120	*
G = 1000	115	125	130	*
NOISE				
Voltage Noise, 1kHz				
R.T.I	$\sqrt{(7nV/\sqrt{Hz})^2 + \left(\frac{90nV/\sqrt{Hz}}{M}\right)^2}$	*	*	*
R.T.O	$\sqrt{(90nV/\sqrt{Hz})^2 + (7nV/\sqrt{Hz} \cdot G)^2}$	*	*	*
TEMPERATURE RANGE				
Specified Performance	0 to +70°C	*	*	–55°C to +125°C
Storage	–65°C to +150°C	*	*	*
POWER SUPPLY				
Power Supply Range	±6V to ±18V	*	*	*
Quiescent Current	3.5mA max	3.5mA	**	**
PACKAGE OPTIONS				
Hermetic 16-Pin Ceramic DIP	D16A	*	*	*
Plastic 16-Pin DIP	N16A	*	*	*

NOTES
*See Section 20 for package outline information.
**Specifications same as AD524J.
***Specifications same as AD524S.
†Specifications subject to change without notice.

APPENDIX C

COMPONENTS PARTS LIST

PARTS LIST for VME8605

ITEM NUMBER	REFERENCE DESIGNATORS	DESCRIPTION
1	U1,2,3,4	IC HS508A
2	U5	IC LM555
3	U6	IC AD524
4	U7	IC HS2425
5	U8	IC 74LS00
6	U9	IC 74LS04
7	U10	IC HS574
8	U11	IC 74LS164
9	U12,13,14	IC 74LS74
10	U15	IC 74S38
11	U16	IC 74LS158
12	U17	IC 74LS139
13	U18	IC 74LS174
14	U19,23,27	IC 74LS244
15	U20	IC 74LS85
16	U21	PAL 12L6
17	U22	PAL 16L8
18	U24	IC 74LS374
19	U25	IC 25LS2521
20	U26	PAL 14L4
21	U28,29	IC 74LS645-1
22	U30	ANALOG DEVICES 949
23	R1	12K OHM RESISTOR
24	R2,4	10K OHM RESISTOR
25	R8,11,12	100 OHM RESISTOR
26	R9	100K OHM RESISTOR
27	R13	4.7K OHM RESISTOR
28	R7,10	100K OHM POTENTIOMETER
29	R5,6	10K OHM POTENTIOMETER
30	R11,12	100 OHM POTENTIOMETER
31	C1,2,26,27	15mf 16V TANTALUM CAPACITOR
32	C3-5,7-11,15-25	.1mf CERAMIC DISC CAPACITOR
33	C6,12	1000 PF CAPACITOR
34	C13,14	1MF CAPACITOR
35	P1	96 PIN MALE DIN CONNECTOR
36	J1	26 PIN MALE CONNECTOR
37	U21,22,26 SOCKET	20 PIN .3" DIP SOCKET
38	U3,4 SOCKET	16 PIN SOCKET
39	U10 SOCKET	28 PIN SOCKET
40	K1,8	3X2 HEADER
41	K2	4X2 PIN HEADER
42	K4,6	3X1 HEADER
43	K9	8X2 HEADER
44	K10	2X7 HEADER
45	K5	1X1 HEADER
46	K7	2X1 HEADER
47	K3	5 PIN HEADER

APPENDIX D

PAL EQUATIONS

```
12L6
860521BCC
REGISTER SELECT LOGIC
MIZAR INC., ST. PAUL, MN.
IIACK DTACK BAS AREADIN RESET BA3 BA2 DS1 BA1 GND
RW AMATCH CONV INTLEN INTEN INTDIS AREAD CHSEL BDS0 VCC


CONV   = /AMATCH*/BAS*RESET*BDS0*/DS1*/BA3*/BA2*/BA1*DTACK*/RW*/IIACK
CHSEL  = /AMATCH*/BAS*RESET*BDS0*/DS1*/BA3*/BA2*BA1*DTACK*/RW*/IIACK
AREAD  = /AMATCH*/BAS*RESET*/BA3*/BA2*BA1*RW*/IIACK
         +/AREADIN*BDS0*/DS1
INTLEN = /AMATCH*/BAS*RESET*BDS0*/DS1*/BA3*BA2*/BA1*DTACK*/RW*/IIACK
INTEN  = /AMATCH*/BAS*RESET*BDS0*/DS1*/BA3*BA2*BA1*DTACK*/RW*/IIACK
INTDIS = /AMATCH*/BAS*RESET*BDS0*/DS1*BA3*/BA2*/BA1*DTACK*/RW*/IIACK
         +/RESET
```

DESCRIPTION: GENERATES LATCH ENABLES FOR CONVERT START, CHANNEL SELECT,
              INTERRUPT ID, AND INTERRUPT ENABLE/DISABLE.
              LOCATED AT U20.

BOARD REVISION C CHANGES: REVISE LAYOUT, MOVE BOARD TO CAD SYSTEM

BOARD REVISION: C   DATE: 7/10/85
PAL   REVISION: A   DATE: 7/10/85

INPUTS:

Pin	Signal	Description	Source
1	IIACK	INVERTED INTERRUPT ACK.	FROM VME BUS - IACK SIGNAL
2	DTACK	DATA ACKNOWLEDGE	FROM U13 - PIN 9
3	BAS	BOARD ADDRESS STROBE	FROM VME BUS - AS SIGNAL
4	AREADIN	ADDRESS READ IN	FROM PAL U20 - PIN 17
5	RESET	RESET	FROM VME BUS - RESET SIGNAL
6	BA3	BOARD ADDRESS 3	FROM VME BUS - ADDRESS SIGNAL
7	BA2	BOARD ADDRESS 2	FROM VME BUS - ADDRESS SIGNAL
8	DS1	DATA STROBE 1	FROM VME BUS - DS1 SIGNAL
9	BA1	BOARD ADDRESS 1	FROM VME BUS - ADDRESS SIGNAL
10	GND	GROUND	
11	RW	READ/WRITE	FROM VME BUS VIA PAL U26-17
12	AMATCH	ADDRESS MATCH	FROM PAL U26 - PIN 16
19	BDS0	BOARD DATA STROBE 0	FROM VME BUS - DS0 SIGNAL
20	VCC	VOLTAGE INPUT	

OUTPUTS:

Pin	Signal	Description	Destination
13	CONV	CONVERSION	TO U8 - PIN 13 INDICATES START OF A/D CONVERSION
14	INTLEN	INTERRUPT LATCH ENABLE	TO U24 - PIN 11 ENABLES INTERRUPT VECTOR ONTO BUS
15	INTEN	INTERRUPT ENABLE	TO U12 - PIN 1
16	INTDIS	INTERRUPT DISABLE	TO U12 - PIN 4 DISABLES INTRPT
17	AREAD	ADDRESS READ	TO PAL U21-PIN4 & U19/U23-1&19 ENABLES DATA BUFFERS
18	CHSEL	CHANNEL SELECT	TO U8 - PIN 12 & U18 - PIN 9 CHOOSES CONVERSION CHANNEL

```
PAL14L4
860526BCC
ADDRESS MODIFIER AND RW DECODER
MIZAR INC., ST. PAUL, MN.
IIACK HAM NC NC AM4 AM3 AM2 AM0 AM1 GND
LWORD WRITE AM5 NC NC AMATCH RW NC NC VCC


RW    = /WRITE*LWORD*/IIACK*AM5*/AM4*AM3*/AM1*AM0*/HAM
AMATCH = LWORD*/IIACK*AM5*/AM4*AM3*/AM1*AM0*/HAM


DESCRIPTION:  ADDRESS MODIFIER DECODER AND R/W DECODER.
              LOCATED U26.


BOARD REVISION C:  REVISE LAYOUT, MOVE BOARD TO CAD SYSTEM


BOARD REVISION:  C    DATE: 7/10/85
PAL   REVISION:  A    DATE: 7/10/85


--------------------------------------------------------------------
```

INPUTS:

Pin	Signal	Description	Source
1	IIACK	INVERTED INT. ACKNOWLED.	FROM VME BUS VIA U9 – PIN 8
2	HAM	HIGH ADDRESS MATCH	FROM U25 – PIN 19 INDICATES VALID A8-A15 ADDRESS
3	NC	NOT CONNECTED	
4	NC	NOT CONNECTED	
5	AM4	ADDRESS MODIFIER 4	FROM VME BUS
6	AM3	ADDRESS MODIFIER 3	FROM VME BUS
7	AM2	ADDRESS MODIFIER 2	FROM VME BUS
8	AM0	ADDRESS MODIFIER 0	FROM VME BUS
9	AM1	ADDRESS MODIFIER 1	FROM VME BUS
10	GND	GROUND	
11	LWORD	LONGWORD	FROM VME BUS-INDICATES 32 BIT XFERS
12	WRITE	WRITE	FROM VME BUS – READ/WRITE LINE
13	AM5	ADDRESS MODIFIER 5	FROM VME BUS
14	NC	NOT CONNECTED	
15	NC	NOT CONNECTED	
18	NC	NOT CONNECTED	
19	NC	NOT CONNECTED	
20	VCC	VOLTAGE INPUT	

```
--------------------------------------------------------------------
```

OUTPUTS:

Pin	Signal	Description	Destination
16	AMATCH	ADDRESS MATCH	TO PAL U21 – PIN 12 INDICATES VALID ADDRESSING FOR THIS BOARD
17	RW	READ/WRITE	TO PALS U21 & U22 PIN 11 DELAYS READ/WRITE LINE FROM BUS

APPENDIX E

SCHEMATIC

MIZAR INC.

VME8605

REV. C

Appendix J -   **Battery Package Data**

List of Batteries

SUPPLIER :

Allied Electronics

MANUFACTURER :

Panasonic

PART NO.	NOMINAL 10 HR. (Ah)	CAPACITY 20 HR. (Ah)	V
LCR12V24P	22	24	12
LCL12V38P	34	38	12
LCR12V3PF	2.8	3.0	12
LCR12V6.5P	6	6.5	12

Wire Color List

WIRE COLOR	PIN NUMBER	BATTERY
blue/blue	1	+12V circuit
black	2	-24V ground
red/red	3	+12V FIP
black	4	ground AOI
orange/orange	5	-12V circuit
white/purple/black	6	+24V gyro
blue	7	+12V AOI
black	8	ground circuit
N/C	9	
black	10	ground circuit
black	11	ground circuit
blue/blue	12	+12V circuit
N/C	13	
black	14	ground FIP
orange	15	-12V circuit
black/black	16	ground circuit

Appendix K -   **List of Parts**

Parts List for All FIP Boards

The Following list is accurate as of 10/9/89.

Display Board:

Connectors:

#	Part	Used For
1	Molex-3	CON5 (Pitch & Roll)
2	Molex-4	CON 3, CON 4 (Magnometer Connectors)
1	Molex-6	POW (Power Connector)
2	Berg-2	TP1, TP2 (Test points for Display Circuits)
2	Berg-3	CON1, CON2 (Display Pots)
1	Berg-50	CON10 (Bus)
2	Berg-20	Alt., AS (Display Connectors)
4	Berg-8	CON6, CON7, CON8, CON9 (Display Jumpers)

Resistor Values:

R1 = 100 K	R2 = 36 K	R3 = 100 K	R4 = 100 K
R5 = 100 K	R6 = 12.5 K	R7 = 100 K	R8 = 100 K
R9 = 2 K	R10 = 2 K	R11 = 2 K	R12 = 10 K
R13 = 10 K	R14 = 10 K	R15 = 10 K	R16 = 2 K
R17 = 2 K	R18 = 2 K	R19 = 10 K	R20 = 10 K
R21 = 10 K	R22 = 10 K		

Totals:

#	Resistor Value
12	10 K
6	100 K
1	36 K
1	12.5 K
2	2K

Capacitor Values:

C1 = 0.44 μF   C2 = 0.44 μF   C3 = 0.13 μF   C4.. = 0.13 μF

Totals:

#	Capacitor Value
2	0.44 μF
2	0.13μF
8	0.1 μF (Bypass. Not listed above)

ICS:

IC1 = MC34082   IC2 = MC34082   IC3 = MC34082

IC4 = MC34082   IC5 = REF-01   IC6 = REF-01
IC7 = REF-01   IC8 = REF-01

Totals:

#	Part
4	MC34082
4	Ref-01

IC Sockets:

#	Part.........	Used For...
8	DIP 8	MC34084 & Ref-01

FIP Big Board:

Connectors and Sockets:

#	Part	Used For
2	MOLEX 3	RP, PWR (Roll/Pitch, Power Connectors)
1	Berg 6	3-Axis (3 Axis)
1	Berg 8	ALT/AS (Altitude/Airspeed)
1	Berg 26	CON26
1	Berg 50	CON50

Resistor Values:

R1 = 10 K	R2 = 10 K	R3 = 28.5K	R4 = 12.5K
R5 = 8.2K	R6 = 17.5K	R7 = 19.2K	R8 = 77.8K
R9 = 10K	R10 = 10K	R11 = 10K	R12 = 10K
R13 = 10 K	R14 = 10 K	R15 = 28.5K	R16 = 12.5K
R17 = 8.2K	R18 = 17.5K	R19 = 19.2K	R20 = 77.8K
R21 = 10K	R22 = 10K	R23 = 10K	R24 = 10K
R25 = 100K	R26 = 10K	R27 = 330K	R28 = 28.5K
R29 = 12.5K	R30 = 8.2K	R31 = 17.5K	R32 = 19.2K
R33 = 77.8K	R34 = 10K	R35 = 10K	R36 = 10K
R37 = 10K	R38 = 100K	R39 = 330K	R40 = 28.5K
R41 = 12.5K	R42 = 8.2K	R43 = 17.5K	R44 = 19.2K
R45 = 77.8K	R46 = 10K	R47 = 10K	R48 = 10K
R49 = 10K	R50 = 100K	R51 = 10K	R52 = 330K
R53 = 28.5K	R54 = 12.5K	R55 = 8.2K	R56 = 17.5K
R57 = 19.2K	R58 = 77.8K	R59 = 10K	R60 = 10K
R61 = 10K	R62 = 10K	R63 = 1 M	R64 = 1 M
R65 = 1 M	R66 = 1 M	R67 = 28.5K	R68 = 12.5K
R69 = 8.2K	R70 = 77.8K	R71 = 19.2K	R72 = 17.5K
R73 = 1K	R74 = 90K	R75 = 1K	R76 = 90K
R77 = 1 M	R78 = 1 M	R79 = 1 M	R80 = 1 M
R81 = 28.5K	R82 = 12.5K	R83 = 8.2K	R84 = 17.5K
R85 = 19.2K	R86 = 77.8K	R87 = 1K	R88 = 200K

R89 = 1K   R90 = 200K   R91 = 10K

Totals:
#	Resistor Value
3	1 K
27	10 K
7	28.5 K
7	12.5 K
7	8.2 K
7	17.5 K
7	19.2 K
7	77.8 K
3	100`K
3	330 K
8	1 M

Capacitor Values:

C1 = 0.1 μF	C2 = 0.1 μF	C3 = 400 nF	C4 = 400 nF
C5 = 400 nF	C6 = 480 pF	C7 = 8000 pF	C8 = 80 nF
C9 = 0.1 μF	C10 = 0.1 μF	C11 = 0.1 μF	C12 = 0.1 μF
C13 = 0.1 μF	C14 = 0.1 μF	C15 = 0.1 μF	C16 = 0.1 μF
C17 = 400 nF	C18 = 400 nF	C19 = 400 nF	C20 = 480 pF
C21 = 8000 pF	C22 = 80 nF	C23 = 0.1 μF	C24 = 0.1 μF
C25 = 0.1 μF	C26 = 0.1 μF	C27 = 0.1 μF	C28 = 0.1 μF
C29 = 0.1 μF	C30 = 0.1 μF	C31 = 400 nF	C32 = 400 nF
C33 = 400 nF	C34 = 480 pF	C35 = 8000 pF	C36 = 80 nF
C37 = 0.1 μF	C38 = 0.1 μF	C39 = 0.1 μF	C40 = 0.1 μF
C41 = 0.1 μF	C42 = 0.1 μF	C43 = 0.1 μF	C44 = 0.1 μF
C45 = 400 nF	C46 = 400 nF	C47 = 400 nF	C48 = 480 pF
C49 = 8000 pF	C50 = 80 nF	C51 = 0.1 μF	C52 = 0.1 μF
C53 = 0.1 μF	C54 = 0.1 μF	C55 = 0.1 μF	C56 = 0.1 μF
C57 = 0.1 μF	C58 = 0.1 μF	C59 = 400 nF	C60 = 400 nF
C61 = 400 nF	C62 = 480 pF	C63 = 8000 pF	C64 = 80 nF
C65 = 0.1 μF	C66 = 0.1 μF	C67 = 0.1 μF	C68 = 0.1 μF
C69 = 0.1 μF	C70 = 0.1 μF	C71 = 0.1 μF	C72 = 400 nF
C73 = 400 nF	C74 = 400 nF	C75 = 480 pF	C76 = 8000 pF
C77 = 80 nF	C78 = 0.1 μF	C79 = 0.1 μF	C80 = 0.1 μF
C81 = 0.1 μF	C82 = 0.1 μF	C83 = 0.1 μF	C84 = 0.1 μF
C85 = 0.1 μF	C86 = 400 nF	C87 = 400 nF	C88 = 400 nF
C89 = 480 pF	C90 = 8000 pF	C91 = 80 nF	C92 = 0.1 μF
C93 = 0.1 μF	C94 = 0.1 μF	C95 = 0.1 μF	C96 = 0.1 μF
C97 = 0.1 μF	C98 = 0.1 μF		

Totals:
#	Capacitor Value
56	0.1 μF
21	400 nF
7	480 pF

7	8000 pF
7	80 nF

POTS:

Pot1 = 10K	Pot2 = 50 K	Pot3 = 50 K	Pot4 = 10K
Pot5 = 50 K	Pot6 = 50 K	Pot7 = 50 K	Pot8 = 50 K
Pot9 = 50 K	Pot10 = 50 K	Pot11 = 50 K	Pot12 = 50 K
Pot13 =50 K	Pot14 = 50 K	Pot15 = 50 K	Pot16 = 50 K

Totals:

#	Part
2	10 K
14	50 K

ICS:

IC1 = OP-77	IC2 = MC34084	IC3 = OP-77
IC4 = OP-77	IC5 = OP-77	IC6 = MC34084
IC7 = OP-77	IC8 = OP-77	IC9 = OP-77
IC10 = MC34084	IC11 = OP-77	IC12 = OP-77
IC13 = OP-77	IC14 = MC34084	IC15= OP-77
IC16 = OP-77	IC17 = OP-77	IC18 = MC34084
IC19 = OP-77	IC20 = OP-77	IC21 = OP-77
IC22 = MC34084	IC23 = OP-77	IC24 = OP-77
IC25 = OP-77	IC26 = MC34084	IC27 = OP-77
IC28 = OP-77		

Totals:

#	Part
21	OP-77
7	MC34084

IC Sockets:

#	Socket	Used For
21	DIP 8	OP-77
7	DIP 14	MC34084

Notes:

Power Distribution Board:

#	Part	Used For
1	MOLEX 3	PWR (For FIP Big Board)
2	MOLEX 6	IN, POW (Battery Pack, Display Board)

| 2 | REF 01 | Transducer +10V Supply |
| 3 | MOLEX 2 | +10V Connector and +24V Gyro Power |

Battery Pack Board:

#	Component
2	VICOR 5V DC to DC converter
1	MOLEX 6 (FIP Power)
1	MOLEX 4 (AOI +12V & -12VPower)
1	MOLEX 2 (AOI +5V Power)
1	UNknown Battery Pack Connector

Appendix L -   VME Specialists SBC-2 68010 Board Manual

Technical Manual

SBC2

Single Board Computer

for the VMEbus

Revision A

First Edition
Copyright 1986 by VMEspecialists

VME SPECIALISTS, INC.
558 Brewster Avenue #1
Redwood City, California 94063 USA

415-364-3328

1204 oBrien Dr
Menlo Park, Ca
94025

2

Table of Contents

3

Table of Figures

4

1. General Product Description

The VMEspecialists SBC2 (figure 1) is a general purpose data processing module fully compatible with the VMEbus and intended to fill application areas such as dedicated machine control and multiprocessor environments which require high functionality density at low cost. The single board computer has the following features:

* 68000/68010 MPU, 10-Mhz clock rate
* Available in 3U (single height) and 6U (double height) form factors
* 512Kbytes dual port zero-wait-state RAM
* Up to 128Kbytes local zero-wait-state EPROM
* Two serial ports with independently programmable baud rates
* 16-bit counter/timer
* Complete on-board VMEbus system controller, may be disabled
* 7 level interrupt handler; Vectored and autovectored modes
* Compliance with VMEbus specification revision "C"
* Front panel RUN and EXTERNAL lamps
* Front panel RESET and ABORT switches

The processor module is constructed on a seven layer printed circuit board. The module employs the zig-zag in-line packaging style of 256Kx1 RAM devices, a pin grid array packaged 68000/68010, and extensive surface mount packaging.

Each module undergoes extensive functional testing to assure high product reliability. A one-year limited-warranty applies.

This series of VMEbus processing modules is exceptionally well suited to applications which require cost effective data processing for the VMEbus, particularly where space and packaging constraints exist. The dual ported memory, accessable by the local processor and by any other VMEbus master, minimizes board count and maximizes memory access rate for systems employing I/O with direct memory access capability.

2. Model Numbers

SBC2
Single board computer, equipped with 10 Mhz 68000,
512Kbytes dual port DRAM, sockets for up to 128Kbytes EPROM.
Configured for EPROM starting at location 000000,
local RAM starting at 080000 (processor address space),
RAM starts at 080000 (VMEbus address space).
3U (single height) front panel. With user's manual.

Add one or more of the following suffixes to specify other configurations:

-010
Provide 10 Mhz 68010 in place of 68000

-AD
Address EPROM at fc0000-fdffff.
DRAM begins at 000000 (processor address space) and at
000000 (VMEbus address space)
On power-up, the SSP and PC are fetched from EPROM.

-6U
Provide 6U (double height) front panel

Figure 1. The SBC2 VMEbus Data Processor

3. Inspection, Warranty, and Repair

Upon receipt, carefully inspect the VMEbus module and shipping container for evidence of damage in shipping. Notify the factory immediately if shipping related damage is suspected.

Limited Warranty

VMEspecialists warrants this product to be free from defects in workmanship and materials under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, VMEspecialists' sole responsibility shall be to repair, or at VMEspecialists' option to replace, the defective product, provided the product is returned transportation prepaid and insured to VMEspecialists. All replaced products become the sole property of VMEspecialists.

VMEspecialists' warranty of and liability for defective products is limited to that set forth above. VMEspecialists disclaims and excludes all other product warranties or product liability, expressed or implied, including, but not limited to, any implied warranties of merchantability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Service Policy

Before returning a product for repair, verify as well as possible that the suspected unit is at fault. Then call the factory for a Return Material Authorization (RMA) number. Carefully package the unit, in the original shipping carton if this is available, and ship prepaid and insured with the RMA number written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. VMEspecialists will not be responsible for damage due to improper packaging of returned items.

Out of Warranty Repairs

Out of warranty repairs will be billed on a material and labor basis. The current minimum repair charge is $100. Customer approval will be obtained before repairing any item if the repair charges will exceed one third of the quantity one list price for that unit. Return transportation and insurance will be billed as part of the repair and is in addition to the minimum charge.

VMEspecialists also makes available repair on an immediate exchange basis. In most cases, a replacement can be shipped on the day of request. This service is billed at a flat rate, currently 30% of the quantity one price.

7

4. Specifications

Processor	10 Mhz 68000 (68010 opt)
EPROM (Capacity/Supplied)	128Kbytes/0
EPROM types	2764, 27128, 27256, 27512
RAM (Capacity/Supplied)	512Kbytes/512Kbytes
RAM type	Dynamic dual ported for access by both the local processor and by the VMEbus.
RAM address configuration	RAM can be partitioned in 64Kbyte increments (processor address space) RAM can be configured to begin on any 512Kbyte boundary (VMEbus space) Address mappings are defined in programmable logic
Wait states (EPROM/RAM)	0/0
Serial ports	Two, RS232-C, using 68681 device
Counter Timer	One 16-bit, using 68681 device
Baud rates	Independently Programmable: 50 to 38.4 Kbaud
VMEbus requester	Any one of R(k), k=0..3 (STAT), RWD
VMEbus Compatibility	Rev. C
Master data transfer options	A24:D16

VMEbus system controller:

Arbiter	Single level (may be disabled)
SYSRESET* driver	Power up or front panle button (may be disabled)
SYSCLOCK driver	(may be disabled)
TOUT	drives BERR* after min 12.6, max 18.9 usec. (may be disabled)
Interrupt handler	Seven total levels (including local) Local sources (Abort button, 68681 SIO/CT)
Physical configuration	SINGLE (opt. avail. with double high panel)
Front panel switches	ABORT, RESET
Front panel lamps	RUN, EXTERNAL
Address space (Standard)	000000 -- 01ffff EPROM (128Kbytes)
	080000 -- 0fffff RAM (512Kbytes)
	100000 -- dfffff VMEbus
	fe0000 -- feffff local 68681 SIO/CT
	ff0000 -- ffffff VMEbus short I/O

```
         (Alternate)      000000 -- 07ffff   RAM     (512Kbytes)
                          080000 -- bfffff   VMEbus
                          fc0000 -- fdffff   EPROM   (128 Kbytes)
                          fe0000 -- feffff   local 68681 SIO/CT
                          ff0000 -- ffffff   VMEbus short I/O
```

Power requirements   +5 VDC, 2.6 Amps typ., 3.6 Amps max.
                         +12 VDC, .035 Amps max.
                         -12 VDC, .035 Amps max.

Operating temperature   0 to +55 degrees C

Storage temperature   -40 to +80 degrees C

Relative humidity   0 to 90% non-condensing

SIZE: SBC2:   129 mm. high, 20 mm. wide, 172 mm. deep
                 SBC2-6U: 262 mm. high, 20 mm. wide, 172 mm. deep
                     (viewed from front panel)

WEIGHT: 0.23 Kg, 0.5 pounds.

5. Installation and Jumper Options

Prior to installation, the module options must be configured by way of jumpers. Options include:

* Specification of the EPROM type
* Enable/Disable control over individual system controller functions
* Assignment of interrupt handler levels
* Specification of VMEbus requester priority level

5.1 EPROM type options

Jumper group A is located below EPROM 2C. There are six posts positioned as indicated below:

```
                              I    2C       I   1C      I
                              I_____   I_____   I

        0    0    0    0    0    0    0
        6    5    4    3    2    1    0
```

Jumper Group A

EPROM TYPE	CONNECT	and	CONNECT
2764	2 -- 3		5 -- 6
27128	2 -- 3		5 -- 6
27256	1 -- 2		5 -- 6
27512	1 -- 2		4 -- 5

We recommend use of 200 ns. access time or faster EPROMS.

The socket at location 1C holds the device driving D0-D7 (ODD ADDRESS). Device 2C drives D8-D15 (EVEN ADDRESS).

5.2 System controller options

The system controller functions may be individually enabled/disabled through jumper block JPRC, having four post pairs. Jumper block C is located just below 3B.

```
                    10    02
                    30    04
                    50    06
                    70    08
```

CONNECT 1--2:   To enable this module to drive SYSRESET* on power-up and when the RESET button on the front panel is pressed.

CONNECT 3--4:   To enable this module to drive SYSCLK, the VMEbus 16 Mhz system clock. Be sure that only one module in your system is driving SYSCLK.

CONNECT 5--6:   To enable the bus timeout watchdog timer on this

module. If no VMEbus slave responds to a transfer request initiated by ANY VMEbus master within a time interval of 12.8 to 19.2 microseconds, the watchdog timer will abort the data transfer cycle with BERR*.

CONNECT 7--8:   To enable the VMEbus arbitor of this module. Only one module in your system can be the system arbitor. If you select this module to perform the arbitor functions, then this module MUST be in slot 1.

5.3 Interrupt handler options

The microprocessor unit recognizes seven distinct interrupt levels, with multiple interrupters permitted on any one level. Jumper block B defines which of the 11 interrupt sources will be active, and the mapping of interrupt sources to MPU interrupt level.

Find jumper block B between 3B and 4B near the board center. Factory default jumper locations are indicated by dashed lines.

```
MPU level 7        2 0 ---- 0 1      ABORT button
                            0 3      VMEbus level 7
MPU level 6        5 0 ---- 0 4      VMEbus level 6
MPU level 5        7 0 ---- 0 6      VMEbus level 5
MPU level 4        9 0 ---- 0 8      VMEbus level 4
MPU level 3       11 0 ---- 0 10     68681 interrupt
MPU level 2       13 0 ---- 0 12     VMEbus level 2
MPU level 1       15 0 ---- 0 14     VMEbus level 1
                            0 16     VMEbus level 3
VMEbus BCLR*      18 0      0 17     VMEbus ACFAIL*
```

Each interrupt level has been defined in the PAL located at position 5B as being either vectored or autovectored.

Default Interrupt Mapping

MPU Level	Vectored/Autovectored
(highest) 7	Autovector
6	Vector
5	Vector
PRIORITY 4	Vector
3	Vector
2	Vector
(lowest) 1	Vector

5.4 VMEbus requester options

Jumper group D defines the priority of the VMEbus requester.   It is
located just above and to the right of P1.

```
15 13 11  9  7  5  3  1
 0  0  0  0  0  0  0  0

    0  0  0  0  0  0  0
   14 12 10  8  6  4  2
```

There are four bus request levels, 0 through 3.   Level 3 has highest
priority.   If you enable the local bus arbitor, or if you are using any
SINGLE level arbitor, then you must use bus request level 3.

To use bus request level 3:   CONNECT 1--2, 3--4, 5--6, 7--9,
 (Factory preset)   8--10, 11-12

```
              0  0  0  0--0  0  0  0
                       I        I  I  I
              0  0  0--0  0  0  0
```

To use bus request level 2:   CONNECT 1--2, 3--4, 5--9, 10--6,
                                           7--8, 12--14

To use bus request level 1:   CONNECT 1--2, 3--9, 4--10, 5--6,
                                           7--8, 12--13

To use bus request level 0:   CONNECT 1--9, 2--10, 3--4, 5--6,
                                           7--8, 12--15

5.5 Installation in a VMEsystem

This module provides continuity of bus grant and interrupt acknowledge
daisy chains. Be sure to check the following prior to installation:

[1] Ensure that the backplane bus grant daisy chain jumpers have been
removed for this slot.

[2]   Ensure that empty slots between slot 1 and this slot have bus
grant and interrupt acknowledge diasy chain jumpers inserted.

[3]   Be sure to install this module in slot 1 if the board's bus
arbitor has been enabled.

[4] Is this slot VMEbus compatible, with VME voltage levels? Nominal
voltages are: +5 VDC,   P1-A32, P1-B32, P1-C32
                +12 VDC,   P1-C31
                -12 VDC,   P1-A31
                GND,   P1-A11, P1-A15, P1-A17, P1-A19, P1-B20, P1-C9

Figure 2.   Jumper Positions

13

6. Address Modifier Codes

During VMEbus data transfer cycles, the module asserts the following address modifier codes:

TRANSFER TYPE	ADDRESS MODIFIER CODE (HEX)
Standard supervisory program access:	3E
Standard supervisory data access:	3D
Standard non-privileged program access:	3A
Standard non-privileged data access:	39

For references within the short I/O space (FF0000 through FFFFFF), the module asserts:

Short supervisory access:	2D
Short non-privileged access:	29

7. VMEbus Interface Signals

P1 Connector Assignments:

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5 STDBY	+12V
32	+5V	+5V	+5V

Notes:

IACKIN* is connected to IACKOUT* on board
The following lines are not used:
SYSFAIL*, SERCLK, SERDAT, +5 STDBY, IRQ7*..1*

8. <u>Serial Connectors JA and JB</u>

<u>Jb Pinout (68681 Channel A)</u>

Pin Number	Name	Direction
1	GND	
3	RXDA	IN
5	TXDA	OUT
7	CTSA	IN
9	RTSA	OUT
13	GND	

<u>Ja Pinout</u>

Pin Number	Name	Direction
1	GND	
3	RXDB	IN
5	TXDB	OUT
7	CTSB	IN
9	RTSB	OUT
13	GND	

9. Parts List

PART NUMBER	DESCRIPTION	LOCATION
1001002125	Data delay line, 125ns	4C
1001002225	Data Delay Device,DDU-222-50	5C
1001100001	LED(green), SPG-5731 REC STANLEY	LED1
1001100002	LED(red),SPR-5731 REC STANLEY	LED2
1001100368	3.68 Mhz Oscillator, TTL Crystal	XTL1
1001101116	16Mhz Oscillator, TTL Crystal	6B
1001101120	20 Mhz Oscillator, TTL Crystal	9C
1001205812	Switch cap, (5081-3, red)	SW2
1001205813	Switch cap (5081-2, black)	SW1
1001210110	Computer switch, (EP11-D1-A-B-E)	SW1,SW2
1002000020	SMS 20 Pin Inline strip socket	13C
1002001124	Socket, 24 pin IC machine screw	2B
1002003926	Molex connector, 39-26-7148	J1,J2
1002004100	Socket, 20 pin IC machine screw	3B,4B,5B,10C,91C,91G
1002006810	Socket,64 pin, T&B Ansley pin grid array	1B
1002012014	Socket, 14 pin sip	1C,2C
1003062260	Capacitor, tantalum, 22ufd	C1
1003064760	Capacitor, tantalum, 47ufd	C2
1003071000	Capacitor, ceramic, 10pfd, .100 rad 10v	C5
1003071010	Capacitor, ceramic, 100pfd, disk	C3,C4
1003073340	Capacitor, ceramic, .1ufd, chip	C23-C45
1004100103	Resistor, 10 kohm chip	R10
1004100202	Resistor, 2 kohm chip	R12
1004100221	Resistor, 220 ohm chip	R14
1004100471	Resistor, 470 ohm chip	R8,R11,R5
1004100472	Resistor, 4.7 kohm chip	R6,R7
1004112470	Resistor, 47 ohm chip	R1-R4,R9,R92
1004704729	Resistor network, Allen Bradley, 710A472	RP1-RP3
1004724704	Resistor network, Allen Bradley, 708B470	RP4,RP5
1005014001	Diode, 1N4001	D1
1006002505	Amp Connector, # 532505-1	P1
1006100025	Header, dualpin x 25	JPRA-D
1007000005	PCB SBC2 0035A	
1007040373	IC 74F373	91F
1007140244	74F244, IC chip	5A,6A,8A,9A
1007140245	74F245, IC chip	1A,2A,91A,91B
1007140257	74F257, IC chip	11C,12C,91D,91E,91F
1007140367	74F367, IC chip	4A
1007420153	IC, 82S153	91C
1008000014	74F14, IC	6C
1008000074	74F74, IC	3C
1008100760	74AS760, IC	3A

PART NUMBER	DESCRIPTION	LOCATION
1008200590	74LS590, IC	8C
1008751488	1488, IC	17D
1008751489	1489, IC	18D
1009001681	Tib Pal 16L8-15CN	3B,4B,5B,10C,91G
1009002010	Pal 20L10A	2B
1009704256	M4256L-12 Zip Ram(120ns)	1D-16D
1009908000	MC 68000R10	1B
1009908681	MC 68681 P	13C
1011003002	3u Front Panel, punched for SBC2	
1011008000	Mounting brackets for front panels	
1011008001	Screw,nut pair, 2.5x10mm	
1012000002	Shipping boxes, M-402	

```
                    PARTNO   0204-0062 ;
                    NAME     INTRFT;
                    DATE     06/05/86 ;
                    REV      02 ;
                    DESIGNER LEHMANN ;
                    COMPANY  VMEspecialists ;
                    ASSEMBLY CPU1 ;
                    LOCATION 4B   ;


/**********************************************************************/
/*  This device performs two independent functions:               */
/*      1) Adr MUX for DRAM A8                                     */
/*      2) Prioity encoder as in 74LS148                          */
/**********************************************************************/
/*  Allowable Target Device Types:      PAL16L8-15                 */
/**********************************************************************/


/** Inputs **/

PIN    1   =  A17     ;      /*  68000 address line A17         */
PIN    2   =  A18     ;      /*                        A18      */
PIN    3   =  ADSEL   ;      /*  ADR MUX Select                 */
PIN    4   = !IRQ7    ;      /*  Highest Priority Int RB        */
PIN    5   = !IRQ6    ;      /*                                 */
PIN    6   = !IRQ5    ;      /*                                 */
PIN    7   = !IRQ4    ;      /*                                 */
PIN    8   = !IRQ3    ;      /*                                 */
PIN    9   = !IRQ2    ;      /*                                 */
PIN    11  = !IRQ1    ;      /*         Lowest Priority         */
PIN    16  = !AS      ;      /*  68000 AS/                      */
PIN    17  = !RES     ;      /*  VMEbus SYSRESET                */


/** Outputs **/

PIN    12  = !IPL0    ;      /*  68000 Interrupt inputs         */
PIN    13  = !IPL1    ;      /*                                 */
PIN    14  = !IPL2    ;      /*                                 */
PIN    15  = !RESET   ;      /*  RESET to board                 */
PIN    18  = !START   ;      /*  START after reset fetch SSP & PC */
PIN    19  = !RA8     ;      /*  DRAM address 8                 */


/** Declarations and Intermediate Variable Definitions **/


/** Logic Equations **/

RA8  =  A17 & ADSEL
      # A18 & !ADSEL ;


IPL2 =     IRQ7 # IRQ6 # IRQ5 # IRQ4 ;


IPL1 =     IRQ7
       #  IRQ6
       #  IRQ3 & !IRQ4 & !IRQ5
       #  IRQ2 & !IRQ4 & !IRQ5 ;


IPL0 =     IRQ7
       #  IRQ5 & !IRQ6
       #  IRQ3 & !IRQ4 & !IRQ6
       #  IRQ1 & !IRQ2 & !IRQ4 & !IRQ6 ;


RESET =    RES ;

START =    RES
       #   START & !AS
       #   START & !A18 ;
```

```
                    PARTNO   0204-0075 ;
                    NAME     TXCTL;
                    DATE     08/30/86 ;
                    REV      01 ;
                    DESIGNER LEHMANN ;
                    COMPANY  VMEspecialists ;
                    ASSEMBLY SBC2 ;
                    LOCATION 916   ;


        /****************************************************************/
        /*                                                            */
        /*                                                            */
        /*                                                            */
        /****************************************************************/
        /* Allowable Target Device Types:      PAL16L8-15           */
        /****************************************************************/


        /** Inputs **/

        PIN   1  = 'DS0    ;      /*   68000 LDS*                  */
        PIN   2  = 'DS1    ;      /*   68000 UDS*                  */
        PIN   3  = 'RAS    ;      /*   Memory RAS/                 */
        PIN   4  = 'CAS1   ;      /*   Memory CAS1/                */
        PIN   5  = 'REFRESH ;     /*   Memory REFRESH/             */
        PIN   6  = 'VDS0   ;      /*   VMEbus DS0*                 */
        PIN   7  = 'VMEDS1 ;      /*   VMEbus DS1* (UNBUFFERED)    */
        PIN   8  = 'CAS0   ;      /*   Memory CAS0/                */
        PIN   9  = 'VME    ;      /*   VME master has memory bus   */
        PIN  11  = 'DRV    ;      /*   This board is VMEbus master */
        PIN  17  = 'VMEWR  ;      /*   VMEbus WRITE*               */


        /** Outputs **/

        PIN  12  = 'DS     ;      /*   Owner of mem bus asserts DS */
        PIN  13  = 'MDS0   ;      /*   Memory bus DS0/             */
        PIN  14  = 'MDS1   ;      /*   Memory bus DS1/             */
        PIN  15  = 'VMEDAT ;      /*   Enable VMEbus transceivers  */
        PIN  16  = 'EXTDAT ;      /*   Enable 68000-mem bus xcvrs  */
        PIN  18  = 'VMEDIR ;      /*   Dir of VMEbus data xcvrs    */
        PIN  19  = 'VDS1   ;      /*   VMEbus DS1* (BUFFERED)      */


        /** Logic Equations **/

        VDS1          = VMEDS1 ;


        DS            = DS0 & DS1 ;



        MDS0     =      VME & VDS0
                   # 'VME & DS0 ;


        MDS1     =      VME & VMEDS1
                   # 'VME & DS1 ;


        VMEDAT   =      RAS & 'REFRESH & VME
                   # DRV
                   # VMEDAT & VDS0
                   # VMEDAT & VMEDS1 ;


        VMEDIR   =      DRV & 'VMEWR
                   # 'DRV & VMEWR ;
```

```
# DRV
# EXTDAT & DS0
# EXTDAT & DS1 ;
```

```
                    PARTNO    0204-0074 ;
                    NAME      AMDECODE ;
                    DATE      08/30/86 ;
                    REV       01 ;
                    DESIGNER  LEHMANN ;
                    COMPANY   VMEspecialists ;
                    ASSEMBLY  SBC2 ;
                    LOCATION  91C  ;


/****************************************************************/
/*                                                            */
/*                                                            */
/*                                                            */
/****************************************************************/
/* Allowable Target Device Types:       82S153A              */
/****************************************************************/


/**  Inputs  **/

PIN     1   =  A23     ;      /*   VMEbus A23                */
PIN     2   =  A22     ;      /*   VMEbus A22                */
PIN     3   =  A21     ;      /*   VMEbus A21                */
PIN     4   =  A20     ;      /*   VMEbus A20                */
PIN     5   =  A19     ;      /*   VMEbus A19                */
PIN     6   =  AM5     ;      /*   VMEbus AM5                */
PIN     7   =  AM4     ;      /*   VMEbus AM4                */
PIN     8   =  AM3     ;      /*   VMEbus AM3                */
PIN     9   =  AM2     ;      /*   VMEbus AM2                */
PIN     11  =  AM1     ;      /*   VMEbus AM1                */
PIN     12  =  AM0     ;      /*   VMEbus AM0                */
PIN     13  = 'DS1     ;      /*   VMEbus DS1*               */
PIN     14  = !DS0     ;      /*   VMEbus DS0*               */
PIN     15  = !BERR    ;      /*   VMEbus BERR*              */
PIN     16  = !IACK    ;      /*   VMEbus IACK*              */
PIN     17  = !DTACK   ;      /*   VMEbus DTACK*             */
PIN     18  =  AS      ;      /*   VMEbus AS*                */

/**  Outputs  **/

PIN     19  = !VRQ     ;      /*  VMEbus requests dualport ram */


/**  Intermediate Logic  **/

SEL     =           !A23 & !A22 & !A21 & !A20 &  A19 & !BERR & !DTACK
                    & !IACK & AS ;

/**  Logic Equations  **/

VRQ     =           AM5 & AM4 & AM3 & !AM1 & AM0 & SEL & DS0
            #       AM5 & AM4 & AM3 & !AM1 & AM0 & SEL & DS1
            #       AM5 & AM4 & AM3 & AM1 & !AM0 & SEL & DS0
            #       AM5 & AM4 & AM3 & AM1 & !AM0 & SEL & DS1
            #       VRQ & AS ;
```

```
                    PARTNO   0204-0074 ;
                    NAME     BSRQSB2 ;
                    DATE     08/30/86 ;
                    REV      01 ;
                    DESIGNER LEHMANN ;
                    COMPANY  VMEspecialists ;
                    ASSEMBLY SBC2 ;
                    LOCATION 2B ;


        /**********************************************************/
        /* This device acts as as SGL level VMEbus arbiter and as a   */
        /* BUS REQUESTER.  It times the address to AS* setup interval.  */
        /*                                                          */
        /**********************************************************/
        /* Allowable Target Device Types:        PAL20L10A        */
        /**********************************************************/


        /** Inputs **/

        PIN    1   =   !BBSY   ;       /*   VMEbus BBSY*            */
        PIN    2   =   !DS0    ;       /*   Local 68000 LDS/        */
        PIN    3   =   !DS1    ;       /*              UDS/         */
        PIN    4   =   !ARBLTCH ;      /*  Output of Request Arbitr Ltch */
        PIN    5   =   !EXT    ;       /*  On board request for VMEbus  */
        PIN    6   =   !BERR   ;       /*  VMEbus BERR*             */
        PIN    7   =   !VMEDS0 ;       /*  VMEbus DS0*              */
        PIN    8   =   !VMEDS1 ;       /*  VMEbus DS1*              */
        PIN    9   =    VMEAS  ;       /*  VMEbus AS*               */
        PIN   10   =   !VMEDT  ;       /*  VMEbus DTACK*            */
        PIN   11   =    BGIN   ;       /*  Bus Grant IN for OUR level  */
        PIN   13   =   !BRXIN  ;       /*  Bus Request IN for our level */
        PIN   21   =    DLYOUT ;       /*  Delay line output         */


        /** Outputs **/

        PIN   14   =    BROUT  ;       /*  Bus Request Output        */
        PIN   15   =   !ARBOUT ;       /*  Bus grant output from arbitor */
        PIN   16   =    BGOUT  ;       /*  Bus grant out from requester */
        PIN   17   =   !EBLAS  ;       /*  Enable Address strobe drive  */
        PIN   18   =   !BERRL  ;       /*  Local BERR/              */
        PIN   19   =    DTACKL ;       /*  Local DTACK/             */
        PIN   20   =   !DLYIN  ;       /*  Delay line input         */
        PIN   22   =    DRV    ;       /*  Address drive enable on VMEbus*/
        PIN   23   =   !EBLTO  ;       /*  Enable Bus Time Out        */


        /** Logic Equations **/



        EBLTO   =   VMEDS0 & !VMEDT
                  # VMEDS1 & !VMEDT ;

        DRV     =   DLYOUT & ARBLTCH & !VMEAS & BGIN & !BGOUT & !VMEDT & !BERR & EXT
                  # DRV & BGIN
                  # DRV & EXT ;

        BERRL   =   DRV & BERR ;

        DTACKL.OE =  DRV & VMEDT ;

        DTACKL  =   'b'1 ;

        EBLAS   =   DRV & !DLYOUT & DS0
                  # DRV & !DLYOUT & DS1
                  # EBLAS & DS0 .
```

```
BGOUT     =   DLYOUT & BGIN & !ARBLTCH
          #   BGOUT & BGIN ;

BROUT     =   EXT & !EBLAS & DS0
          #   EXT & !EBLAS & DS1 ;

ARBOUT    =   BRIIN & 'BBSY ;

DLYIN     =   BGIN & 'DRV ;
```

```
                        PARTNO    0204-0073 ;
                        NAME      DPRAM    ;
                        DATE      08/30/86 ;
                        REV       01 ;
                        DESIGNER  LEHMANN ;
                        COMPANY   VMEspecialists ;
                        ASSEMBLY  SBC2 ;
                        LOCATION  10C ;


        /************************************************************/
        /* This device control the local DRAM.  There are 512Kbytes of  */
        /* Zero wait state memory (1 bank of 256Kx16).  Asynchronous   */
        /* arbitration is between the refresh timer and the 68000.     */
        /* Arbitration and address decoding are overlapped. Refresh uses */
        /* the CAS before RAS cycle and automatic REFRESH addrss generation  */
        /************************************************************/
        /* Allowable Target Device Types:      PAL16L8-15           */
        /************************************************************/


        /** Inputs **/

        PIN   1    = !ASL       ;     /* 68000's AS/ latched in arbit */
        PIN   2    = !MDS0      ;     /* Memory bus DS0/              */
        PIN   3    = !MDS1      ;     /* Memory bus DS1/              */
        PIN   4    = !RAM       ;     /* RAM address presented 68000  */
        PIN   5    = !DEL40     ;     /* RAS/ delayed 40 ns.          */
        PIN   6    = !DEL100    ;     /* RAS/ delayed 100 ns.         */
        PIN   7    = !ARBTO     ;     /* ARB delayed for metastables  */
        PIN   8    = !VMERQ     ;     /* VME request of ram, latched  */
        PIN   9    =  RFSHTO    ;     /* Refresh timeout              */
        PIN   11   = !VMEDEL    ;     /* VME delayed                  */


        /** Outputs **/

        PIN   12   = !VME       ;     /* Set MUX to VMEbus access     */
        PIN   13   = !VMEDT     ;     /* DTACK* VMEbus                */
        PIN   14   = !REFRESH ;       /* Refresh cycle                */
        PIN   15   = !MDTACK    ;     /* Memory DTACK/, 68000 access  */
        PIN   16   = !CAS1      ;     /* CAS for D0..7                */
        PIN   17   = !CAS0      ;     /* CAS for D8..15               */
        PIN   18   = !RAS       ;     /* RAS/                         */
        PIN   19   = !ARB       ;     /*     ARBITRATE CLOCK          */


        /** Logic Equations **/

        ARB      = !RAS & RAM & !MDTACK & !REFRESH & !VMEDT
                 # VMERQ & !RAS & !MDTACK & !REFRESH & !VMEDT
                 # RFSHTO & !RAS & !MDTACK & !REFRESH & !VMEDT ;


        REFRESH =  ARBTO & !RAS & !DEL40 & !CAS0 & !CAS1 & RFSHTO & !DEL100
                 # REFRESH & !DEL100
                 # REFRESH & DEL40 ;


        MDTACK  =  RAS & !REFRESH & MDS1 & !VMEDEL
                 # RAS & !REFRESH & MDS0 & !VMEDEL
                 # MDTACK & RAM & MDS0
                 # MDTACK & RAM & MDS1 ;


        VMEDT   =  RAS & !REFRESH & VMEDEL & CAS1 & DEL100
                 # RAS & !REFRESH & VMEDEL & CAS0 & DEL100
                 # VMEDT & MDS0
                 # VMEDT & MDS1 ;

        CAS1    =  RAS & !REFRESH & DEL40 & MDS0
```

```
                # CAS1 & ASL & !REFRESH & RAM & MDS0 & MDTACK & !VMEDEL
                # CAS1 & MDS0 & VMEDEL & !REFRESH ;

CAS0    =   RAS & !REFRESH & DEL40 & MDS1
            # REFRESH & !DEL100 & !ARBTO
            # CAS0 & ASL & !REFRESH & RAM & MDS1 & MDTACK & !VMEDEL
            # CAS0 & MDS1 & VMEDEL & !REFRESH ;

RAS     =   REFRESH & !RFSHTO & CAS0
            # REFRESH & !RFSHTO & CAS1
            # ARBTO & !RFSHTO & !VMERQ & !VMEDEL & ASL & RAM & !DEL40
                & !DEL100 & !CAS0 & !CAS1
            # ARBTO & !RFSHTO & VMERQ & VMEDEL & !DEL40 & !DEL100 &
                !CAS0 & !CAS1
            # RAS & !DEL100 & !REFRESH
            # RAS & RAM & !REFRESH & !VMEDEL
            # RAS & VMERQ & !REFRESH & VMEDEL ;

VME     =   ARBTO & !RFSHTO & VMERQ & !RAS & !CAS0 & !CAS1 & !REFRESH
            # VMEDEL & VMERQ
            # VMEDEL & CAS0
            # VMEDEL & CAS1 ;
```

```
                    PARTNO    0204-0073 ;
                    NAME      ADRSB2;
                    DATE      09/01/86 ;
                    REV       01 ;
                    DESIGNER  LEHMANN ;
                    COMPANY   VMEspecialists ;
                    ASSEMBLY  SBC2  ;
                    LOCATION  3B ;


        /******************************************************************/
        /*  This device does address decoding as follows:              */
        /*                                                             */
        /*      ADDRESS:        DESTINATION:                           */
        /*      000000 - 01ffff  ROM       (128 Kbytes)               */
        /*      080000 - 0fffff  RAM       (512 Kbtyes)               */
        /*      100000 - dfffff  VMEbus                               */
        /*      fe0000 - feffff  Local 68681  SIO and counter/timer   */
        /*      ff0000 - ffffff  VMEbus short I/O                     */
        /******************************************************************/
        /* Allowable Target Device Types:   PAL16L8-15                */
        /******************************************************************/


        /** Inputs **/

        PIN [1..8]  = [A23..16] ;      /*  68000 adr 23..16             */
        PIN   9     = !DS    ;         /*  68000 DS/                    */
        PIN   11    = !START ;         /*  Inital SSP and PC fetches    */
        PIN   13    = !AS    ;         /*          AS/                  */
        PIN   17    = !IACK  ;         /*  FC0..2 is interrupt ack      */
        PIN   18    = !EXTIACK ;       /*  interrupt ack for VMEbus     */


        /** Outputs **/

        PIN   12   = !ROM    ;         /*  Local ROM access            */
        PIN   14   = !RAM    ;         /*  Local RAM access            */
        PIN   15   = !SIO    ;         /*  68681 access                */
        PIN   16   = !EXTERNAL ;       /*  VMEbus access               */
        PIN   19   = !SHORT  ;         /*  VMEbus I/O access           */

        /** Declarations and Intermediate Variable Definitions **/

        FIELD ADR = [A23..16] ;

        /** Logic Equations **/


        SHORT  =   A23 & A22 & A21 & A20 & A19 & A18 & A17 & A16 ;

        SIO    =   A23 & A22 & A21 & A20 & A19 & A18 & A17 & !A16 & AS & DS & !IACK ;

        ROM    =   !A23 & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & AS & DS & !IACK ;

        RAM    =   !A23 & !A22 & !A21 & !A20 & A19 & AS & !IACK ;

        EXTERNAL = !A23 & !A22 & !A21 & A20 & AS & !IACK
                 # !A23 & !A22 & A21 & AS & !IACK
                 # !A23 & A22 & AS & !IACK
                 # A23 & !A22 & AS & !IACK
                 # A23 & A22 & !A21 & AS & !IACK
                 # EXTIACK & AS
                 # A23 & A22 & A21 & A20 & A19 & A18 & A17 & A16 & AS & !IACK ;
```

```
                    PARTNO   0204-0085 ;
                    NAME     FCNDECOD ;
                    DATE     02/10/86 ;
                    REV      01 ;
                    DESIGNER LEHMANN ;
                    COMPANY  VMEspecialists ;
                    ASSEMBLY CPU1 ;
                    LOCATION 5B ;


/**********************************************************************/
/* This device decodes the FC2..0 lines of the 68000 to              */
/* chk for interrupt ack cycles, internal or external.  It chooses   */
/* vectored or autovectored IACK cycle, and also includes a BUS TIMOUT ctr */
/**********************************************************************/
/* Allowable Target Device Types:          PAL16L8-15                */
/**********************************************************************/


/** Inputs **/

PIN  [1..3]  = [A1..3] ;       /*  68000 Address A1 to A3       */
PIN  [4..6]  = [FC0..2] ;      /*  68000 Function FC0..2        */
PIN   7      = !AS ;           /*  68000 AS/                    */
PIN   8      = SLOCLK ;        /*  REGULAR CLK 156 Khz          */
PIN   9      = !EBLTO ;        /*  Enable Bus Time Out          */


/** Outputs **/

PIN   12     = !LIACK ;        /*  Local interrupt ACK/         */
PIN   13     = !IACK ;         /*  Any  interrupt ACK/          */
PIN   14     =  STATE1 ;       /*  BTO state machine state1     */
PIN   15     =  STATE0 ;       /*  BTO state machine state0     */
PIN   17     = !BTO ;          /*  Bus Time-out                 */
PIN   18     = !VPA ;          /*  68000 VPA/                   */
PIN   19     = !EXTIACK ;      /*  External interrupt ACK/      */


/** Declarations and Intermediate Variable Definitions **/

FIELD FCN= [FC2..0] ;
FIELD ADR= [A3..1] ;


/** Logic Equations **/

BTO  =  EBLTO & STATE1 & !STATE0 ;

IACK =  FCN:7 ;

LIACK = FCN:7 & AS & !A3 & A2 & A1;

EXTIACK = FCN:7 & A3 & !A2
        # FCN:7 & A3 & A2 & !A1
        # FCN:7 & !A3 & A2 & !A1
        # FCN:7 & !A3 & !A2 & A1 ;

VPA  =  FCN:7 & AS & A3 & A2 & A1 ;

STATE1 =  EBLTO & !STATE1 & STATE0 & !SLOCLK
        # EBLTO & STATE1 ;

STATE0 =  EBLTO & !STATE1 & !STATE0 & SLOCLK
        # EBLTO & !STATE1 & STATE0
        # EBLTO & STATE1 & STATE0 & !SLOCLK ;
```

```
        CHIP.AS      U234 UUU2 ;
        NAME     INTRPT;
        DATE     06/05/86 ;
        REV      02 ;
        DESIGNER LEHMANN ;
        COMPANY  VMEspecialists ;
        ASSEMBLY CPU1 ;
        LOCATION 4B   ;


/*******************************************************************/
/*  This device performs two independent functions:            */
;*      1) Adr MUX for DRAM A8                                  */
/*      2) Prioity encoder as in 74LS148                        */
/*******************************************************************/
/*  Allowable Target Device Types:        PAL16L8-15           */
/*******************************************************************/


/**  Inputs  **/

PIN    1   =   A17    ;      /*   68000 address line A17        */
PIN    2   =   A18    ;      /*                       A18       */
PIN    3   =   ADSEL  ;      /*   ADR MUX Select                */
PIN    4   =  !IRQ7   ;      /*   Highest Priority Int RQ       */
PIN    5   =  !IRQ6   ;      /*                                 */
PIN    6   =  !IRQ5   ;      /*                                 */
PIN    7   =  !IRQ4   ;      /*                                 */
PIN    8   =  !IRQ3   ;      /*                                 */
PIN    9   =  !IRQ2   ;      /*                                 */
PIN   11   =  !IRQ1   ;      /*        Lowest Priority          */
PIN   16   =  !AS     ;      /*   68000 AS/                     */
PIN   17   =  !RES    ;      /*   VMEbus SYSRESET               */


/**  Outputs  **/

PIN   12   =  !IPL0   ;      /*   68000 Interrupt inputs        */
PIN   13   =  !IPL1   ;      /*                                 */
PIN   14   =  !IPL2   ;      /*                                 */
PIN   15   =  !RESET  ;      /*  RESET to board                 */
PIN   18   =  !START  ;      /*  START after reset fetch SSP & PC */
PIN   19   =  !RA8    ;      /*  DRAM address 8                 */


/**  Declarations and Intermediate Variable Definitions  **/

/**  Logic Equations  **/

RA8    =  A17 & ADSEL
       #  A18 & !ADSEL ;


IPL2  =      IRQ7 # IRQ6 # IRQ5 # IRQ4 ;


IPL1  =      IRQ7
        #    IRQ6
        #    IRQ3 & !IRQ4 & !IRQ5
        #    IRQ2 & !IRQ4 & !IRQ5 ;


IPL0  =      IRQ7
        #    IRQ5 & !IRQ6
        #    IRQ3 & !IRQ4 & !IRQ6
        #    IRQ1 & !IRQ2 & !IRQ4 & !IRQ6 ;


RESET =      RES ;

START =      RES
        #    START & !AS
        #    START & !A19 ;
```

P1 CONNECTOR PINOUT

VME SPECIALISTS, INC.
350 BREWSTER AVENUE SUITE #1
REDWOOD CITY, CA 94063

SBC2 VMEBUS PROCESSOR

VME SPECIALISTS, INC.
550 BREWSTER AVENUE SUITE #1
REDWOOD CITY, CA 94063

SBC2 VMEBUS PROCESSOR

VME SPECIALISTS, INC.
550 BREWSTER AVENUE SUITE # 1
REDWOOD CITY, CA 94063

SBC2 VMEBUS PROCESSOR

REV 1A
130-0001-001
SHEET 4 /14

SBC2 VMEBUS PROCESSOR

VME SPECIALISTS, INC.
550 BREWSTER AVENUE SUITE #1
REDWOOD CITY, CA 94063

DATE: SEPTEMBER 9, 1986   SHEET 5 OF 7

VME SPECIALISTS, INC.
550 BREWSTER AVE. REDWOOD CITY, CA

SBC2 VMEBUS PROCESSOR

| D | | 150-0001-001 | REV 1A |
| DATE: | | SHEET 6 OF |

MICROPROCESSOR

VME SPECIALISTS, INC.
550 BREWSTER AVE. REDWOOD CITY, CA

SBC2 VMEBUS PROCESSOR

150-0001-001   REV 1A
SHEET 7 OF

FIP Dump Table Description   11/5/89

Column	Description
0	Air Speed
1	Angle of Attack - always 0
2	Vertical velocity - 0 to 4096 with 2048=0 velocity
3	Heading Deviation
4	Roll Angle
5	Pitch Angle
6	Altitude Deviation

Column	Channel	Description
7	0	Raw Altitude Set
8	1	Raw Heading Set
9	2	0
10	3	0
11	4	Raw Altitude
12	5	Raw Air Speed
13	6	Raw Pitch
14	7	Raw Roll
15	8	Raw Z
16	9	Raw Y
17	10	Raw X

APPENDIX C

VMEbus BACKPLANE CONNECTORS AND VME BOARD CONNECTORS

INTRODUCTION

This appendix identifies the VMEbus backplane J1/P1 connector pin assignments. The following table lists the pin assignments by pin number order. (The connector consists of three rows of pins labeled rows A, B, and C.)

J1/P1 Pin Assignments

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND -	BG2OUT*	GND -
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND -	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND -	BR3*	A23
16	DTACK*	AM0	A22
17	GND -	AM1	A21
18	AS*	AM2	A20
19	GND -	AM3	A19
20	IACK*	GND -	A18
21	IACKIN*	SERCLK (1)	A17
22	IACKOUT*	SERDAT (1)	A16
23	AM4	GND -	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

NOTE:

(1) SERCLK and SERDAT represent provision for a special serial communication bus protocol still being finalized.

APPENDIX B

VMEbus CONNECTOR/PIN DESCRIPTION

INTRODUCTION

This appendix describes the VMEbus pin connections.   The following table
identifies the VMEbus signals by signal mnemonic, connector and pin number, and
signal characteristic.

VMEbus Signal Identification

SIGNAL MNEMONIC	CONNECTOR AND PIN NUMBER	SIGNAL NAME AND DESCRIPTION
ACFAIL*	1B: 3	AC FAILURE - Open-collector driven signal which indicates that the AC input to the power supply is no longer being provided or that the required input voltage levels are not being met.
IACKIN*	1A: 21	INTERRUPT ACKNOWLEDGE IN - Totem-pole driven signal. IACKIN* and IACKOUT* signals form a daisy-chained acknowledge. The IACKIN* signal indicates to the VME board that an acknowledge cycle is in progress.
IACKOUT*	1A: 22	INTERRUPT ACKNOWLEDGE OUT - Totem-pole driven signal. IACKIN* and IACKOUT* signals form a daisy-chained acknowledge. The IACKOUT* signal indicates to the next board that an acknowledge cycle is in progress.
AM0-AM5	1A: 23 1B: 16,17, 18,19 1C: 14	ADDRESS MODIFIER (bits 0-5) - Three-state driven lines that provide additional information about the address bus, such as size, cycle type, and/or DTB master identification.
AS*	1A: 18	ADDRESS STROBE - Three-state driven signal that indicates a valid address is on the address bus.
A01-A23	1A: 24-30 1C: 15-30	ADDRESS bus (bits 1-23) - Three-state driven address lines that specify a memory address.
A24-A31	2B: 4-11	ADDRESS bus (bits 24-31) - Three-state driven bus expansion address lines.
BBSY*	1B: 1	BUS BUSY - Open-collector driven signal generated by the current DTB master to indicate that it is using the bus.
BCLR*	1B: 2	BUS CLEAR - Totem-pole driven signal generated by the bus arbitrator to request release by the current DTB master in the event that a higher level is requesting the bus.

B-1

SIGNAL MNEMONIC	CONNECTOR AND PIN NUMBER	SIGNAL NAME AND DESCRIPTION
BERR*	1C: 11	BUS ERROR - Open-collector driven signal generated by a slave. This signal indicates that an unrecoverable error has occurred and the bus cycle must be aborted.
BG0IN*-BG3IN*	1B: 4,6, 8,10	BUS GRANT (0-3) IN - Totem-pole driven signals generated by the Arbiter or Requesters. Bus grant in and out signals form a daisy-chained bus grant. The bus grant in signal indicates to this board that it may become the next bus master.
BG0OUT*-BG3OUT*	1B: 5,7, 9,11	BUS GRANT (0-3) OUT - Totem-pole driven signals generated by Requesters. Bus grant in and out signals form a daisy-chained bus grant. The bus grant out signal indicates to the next board that it may become the next bus master.
BR0*-BR3*	1B: 12-15	BUS REQUEST (0-3) - Open-collector driven signals generated by Requesters. These signals indicate that a DTB master in the daisy-chain requires access to the bus.
DS0*	1A: 13	DATA STROBE 0 - Three-state driven signal that indicates during byte and word transfers that a data transfer will occur on data bus lines (D00-D07).
DS1*	1A: 12	DATA STROBE 1 - Three-state driven signal that indicates during byte and word transfers that a data transfer will occur on data bus lines (D08-D15).
DTACK*	1A: 16	DATA TRANSFER ACKNOWLEDGE - Open-collector driven signal generated by a DTB slave. The falling edge of this signal indicates that valid data is available on the data bus during a read cycle, or that data has been accepted from the data bus during a write cycle.
D00-D15	1A: 1-8 1C: 1-8	DATA BUS (bits 0-15) - Three-state driven bidirectional data lines that provide a data path between the DTB master and slave.
D16-D31	2B: 14-21 2B: 23-30	DATA BUS (bits 16-31) - Three-state driven bidirectional lines for data bus expansion.
GND	1A: 9,11, 15,17,19 1B: 20,23 1C: 9 2B: 2,12, 22,31	GROUND

SIGNAL MNEMONIC	CONNECTOR AND PIN NUMBER	SIGNAL NAME AND DESCRIPTION
IACK*	1A: 20	INTERRUPT ACKNOWLEDGE – Open-collector or Three-state driven signal from any MASTER processing an interrupt request. Routed via backplane to Slot 1, where it is looped back to become Slot 1 IACKIN* to start the interrupt acknowledge daisy-chain.
IRQ1*-IRQ7*	1B: 24-30	INTERRUPT REQUEST (1-7) – Open-collector driven signals, generated by an interrupter, which carry prioritized interrupt requests. Level seven is the highest priority.
LWORD*	1C: 13	LONGWORD – Three-state driven signal to indicate that the current transfer is a 32-bit transfer.
[RESERVED]	2B: 3	RESERVED – Signal line reserved for future VMEbus enhancements. This line must not be used.
SERCLK	1B: 21	A reserved signal which will be used as the clock for a serial communication bus protocol which is still being finalized.
SERDAT	1B: 22	A reserved signal which will be used as the transmission line for serial communication bus messages.
SYSCLK	1A: 10	SYSTEM CLOCK – A constant 16-MHz clock signal that is independent of processor speed or timing. This signal is used for general system timing use.
SYSFAIL*	1C: 10	SYSTEM FAIL – Open-collector driven signal that indicates that a failure has occurred in the system. This signal may be generated by any module on the VMEbus.
SYSRESET*	1C: 12	SYSTEM RESET – Open-collector driven signal which, when low, will cause the system to be reset.
WRITE*	1A: 14	WRITE – Three-state driven signal that specifies the data transfer cycle in progress to be either read or write. A high level indicates a read operation; a low level indicates a write operation.

SIGNAL MNEMONIC	CONNECTOR AND PIN NUMBER	SIGNAL NAME AND DESCRIPTION
+5V STDBY	1B: 31	+5 Vdc STANDBY – This line supplies +5 Vdc to devices requiring battery backup.
+5V	1A: 32 1B: 32 1C: 32 2B: 1,13,32	+5 Vdc Power – Used by system logic circuits.
+12V	1C: 31	+12 Vdc Power – Used by system logic circuits.
-12V	1A: 31	-12 Vdc Power – Used by system logic circuits.

Appendix M -   VME Specialists VME-750 Board Manual

Technical Manual

VME750

Multi-Function Accessory Module

for the SBC2 VMEbus Computer Module

Revision A

Table of Contents

Table of Figures

1. General Product Description

The VMEspecialists VME750 (figure 1) is a multi-function plug-in accessory module for use with the SBC2 VMEbus computer, having the following features:

* Adds two additional async serial ports with baud rate generators
* Adds an additional 16-bit counter/timer
* Adds a real-time clock with lithium battery power source
* Adds two static memory sockets, which can hold EPROM'S or RAM's. These sockets are also powered by the continuous lithium power source and can be used for up to 64Kbytes of non-volatile RAM, or up to 128Kbytes of EPROM. With the EPROM sockets on the SBC2, a total of 256Kbytes of EPROM can be accommodated. When used with RAM, a front-panel mounted write-protect switch provides a mechanism for insuring data integrity during system operation.
* Provides a socket for an MC68881 floating point coprocessor.
* Provides a socket for an MC68451 memory management unit.
* Provides a four bit parallel input port through which the SBC2 can sense a jumper group for configuration information.

The VME750 mounts rigidly to the SBC2; together they share a double wide front panel. The two board set occupies two positions in the VMEbus.

VME750-C: Without memory management, Without co-processor
VME750-F: With floating point co-processor, without MMU
VME750-M: With memory management unit, without coprocessor
VME750-E: With memory management, with coprocessor

Add -6U suffix to denote 6U (Double High) front panel for secure mounting in 6U chassis.

5

Figure 1. The VME750 Multifunction Accessory

2. Inspection, Warranty, and Repair

Upon receipt, carefully inspect the VME750 module and shipping container for evidence of damage in shipping. Notify the factory immediately if shipping related damage is suspected.

Limited Warranty

VMEspecialists warrants this product to be free from defects in workmanship and materials under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, VMEspecialists' sole responsibility shall be to repair, or at VMEspecialists' option to replace, the defective product, provided the product is returned transportation prepaid and insured to VMEspecialists. All replaced products become the sole property of VMEspecialists.

VMEspecialists' warranty of and liability for defective products is limited to that set forth above. VMEspecialists disclaims and excludes all other product warranties or product liability, expressed or implied, including, but not limited to, any implied warranties of merchantability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Service Policy

Before returning a product for repair, verify as well as possible that the suspected unit is at fault. Then call the factory for a Return Material Authorization (RMA) number. Carefully package the unit, in the original shipping carton if this is available, and ship prepaid and insured with the RMA number written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. VMEspecialists will not be responsible for damage due to improper packaging of returned items.

Out of Warranty Repairs

Out of warranty repairs will be billed on a material and labor basis. The current minimum repair charge is $100. Customer approval will be obtained before repairing any item if the repair charges will exceed one third of the quantity one list price for that unit. Return transportation and insurance will be billed as part of the repair and is in addition to the minimum charge.

VMEspecialists also makes available repair on an immediate exchange basis. In most cases, a replacement can be shipped on the day of request. This service is billed at a flat rate, currently 30% of the quantity one price.

7

3. Specifications

VME compatibility:	Compliance with VMEbus specification rev. "C.1" Shares double wide front panel with SBC2, to form single high (3U) two board set. Provides continuity of IACK and BGOUT daisy chains
Static Memory Capacity:	128Kbytes (EPROM) 64Kbytes (RAM) 200 ns. access time Write protect switch on front panel Uses lithium cell for non-volatility
Serial I/O Ports:	Two async, using MC68681, with integral counter/ timer and dual independently programmable baud rate generators.
Real time clock:	100 year calendar, 12 and 24 hour modes
Optional functions:	MC68881 floating point co-processor MC68451 memory management
Environmental:	Operating temperature: 0 to 55 degrees C Storage temperature: -40 to 80 degrees C Operating humidity: 0 to 90% (no condensation) Storage humidity: 0 to 90% (no condensation)
Power Requirements: (In addition to SBC2 specification)	1.2A MAX (0.9A TYP) at +5VDC +/- 5% 0.035A MAX at +12VDC 0.035A MAX at -12VDC With MC68451 MMU add 0.3A MAX With MC68881 CO-PROC add 0.15A MAX
SIZE:	VME620: 129 mm. high, 20 mm. wide, 172 mm. deep VME620-6U: 262 mm. high, 20 mm. wide, 172 mm. deep (viewed from front panel)
WEIGHT:	0.23 Kg, 0.5 pounds.

4. Installation and Jumper Options

Prior to installation, the board options must be configured by way of jumpers. The ten user configurable jumpers specify :

JPR1: Address strobe timing
JPR2: LDS data strobe timing
JPR3: UDS data strobe timing
JPR4: Static memory type select
JPR5: Static memory type select
JPR6: Static memory type select
JPR7-JPR10: Parallel sense block for configuration information

Refer to figure 2 for assistance in locating jumper block positions.

4.1 Memory management options: Jumper groups 1-3

```
O A
O B   JP1
O C
                    When using the memory management option of
                    VME750, connect A--B on each group JP1..JP3.
O A                 You must jumper A--B when the 68451 MMU
O B   JP2           is in place.
O C
                    When there is no 68451 memory management unit
O A                 in place, connect B--C on each group.
O B   JP3
O C
```

4.2 Static memory type select

Use jumper groups JP4..JP6 to define the type of static memory (if any) in use at locations U12 and U13. Refer to the table below:

Device	JP4	JP5	JP6
2764 EPROM	NONE	B--A	B--A
27128 EPROM	B--C	B--A	B--A
27256 EPROM	B--C	B--A	B--D
27512 EPROM	B--C	B--D	B--D
8Kx8 RAM	A--B	NONE	B--C
32Kx8 RAM	B--C	B--C	B--C

The two static memory devices in the sockets U12 and U13 must be of identical type. The odd addressed device (D07..D00) should be in location U13. The even addressed device (D15..D08) should be in U12. Use 200ns or faster devices.

These options are set at the factory for 8Kx8 static RAMS.

9

4.3 Parallel sense jumper block

Use jumper groups JP7..JP10 to convey configuration information to the processor.   JP7 and JP8 A..C are located beside static memory socket U13. JP9 and JP10 A..C are located on the board edge beside the MC68881 socket.

In each case, jumper A--B to select a data value of 1.   Jumper B--C to select a value of 0.

The jumper blocks can be sensed by the MPU through a read of the 68681 DUART.

Jumper group	DUART Input Pin
JP7	IP2
JP8	IP3
JP9	IP5
JP10	IP4

Figure 2.   Jumper Group Locations

11

4.4 Installation in a VMEsystem

The VME750 accessory module provides continuity of all bus grant and interrupt acknowledge daisy chains. There is no need to jumper these positions on the VMEbus P1 backplane.

This product has been designed to operate in a VMEbus electrical environment. Insertion into any card slot which is not VMEbus compatible is likely to cause serious damage. Please exercise particular care with the 3U sized version of this product, which can be easily damaged if inserted into an I/O slot, rather than into a standard VMEbus P1 slot.

5. Theory of Operation

The VME750 uses the following locations in the SBC2 processor's address space:

F20000 -- F3FFFF	National MM58274 Real Time Clock
F40000 -- F5FFFF	Motorola MC68681 DUART
F60000 -- F7FFFF	Motorola MC68881 Floating Point Processor
F80000 -- F9FFFF	Motorola MC68451 Memory Management Unit
FA0000 -- FBFFFF	Static Memory Sockets U12, U13

5.1. Real-Time Clock

The real-time clock can be accessed through byte transfers to the following addresses:

F20001	Control Register	Split R/W
F20003	10th's of second	Read ONLY
F20005	Seconds	R/W
F20007	Tens of seconds	R/W
F20009	Minutes	R/W
F2000B	Tens of minutes	R/W
F2000D	Hours	R/W
F2000F	Tens of hours	R/W
F20011	Days	R/W
F20013	Tens of days	R/W
F20015	Months	R/W
F20017	Tens of months	R/W
F20019	Years	R/W
F2001B	Tens of years	R/W
F2001D	Day of the week	R/W
F2001F	Clock setting/interrupt register	R/W

The MM58274 can count to 100 years and fully accounts for leap years. It can work in 12 or 24 hour mode.

The VME750 does not support interrupts from the real-time clock. In all reads and writes, only bits D3..D0 are meaningful. Refer to the Ntl MM58274 data sheet for additional information.

5.2. MC68681 DUART

The on-board 68681 provides two asyncronous serial channels, a counter/timer, and two programmable baud rate generators. The device can interrupt on processor interrupt level 1 (autovector). The 68681 is also used as a parallel input port to sense the configuration of jumper blocks JP7 through JP10.

Access the 68681 through byte reads/writes to the odd addresses in the range F40001 through F4001F.

5.3. MC68881 Floating Point Co-processor

There is a socket provided for an optional MC68881 device. Software can attempt a read or write to one of the MC68881 locations (mapped F60000 to F7FFFF). If a BERR results, then the 68881 option is not available (the socket is empty). The user needs only insert an MC68881 into the socket at lcoation U11 to activate this option.

5.4. MC68451 Memory Management Unit

The VME750 is designed to work with or without memory management. Memory management will slow processor bus cycles, but is often a requirement for UNIX(TM:AT&T) and other multi-user general purpose systems. The 68451 maps logical to physical addresses and permits segment write protection. The 68451 is mapped into the processor's address space in the range of F80000 through F9FFFF.

The socket at location U3 MUST contain either an MC68451 or a special pass-through circuit module (when the MMU option is not required). Jumper groups JP1 through JP3 must be set consistent with the contents of socket U3.

5.5. Static Memory (Sockets U12, U13)

Data transfers to or from addresses FA0000 through FBFFFF are directed to the static memory sockets U12 and U13. Jumper groups J4..JP6 should be set according to the two identical static memory devices in place.

There is a front panel mounted switch which can be set to either a R/W position or to a W-PROT position for write protection of static RAM. The odd addressed device (D07..D00) is held in socket U13. The even addressed device (D15..D08) is held in socket U12. The devices should have 200 ns. or faster access time.

The static memory is powered by the on-board lithium battery for non-volatile operation of static RAM. Be sure to use the low power versions of static memory devices, designed for battery backup.

The two sockets, U12 and U13 can hold EPROM devices 2764, 27128, 27256, or 27512. They can hold 8Kx8 or 32Kx8 static RAMs.

13

6. Serial Connectors JA and JB

Ja Pinout (68681 Channel A)

Pin Number	Name	Direction
1	GND	
3	RXDA	IN
5	TXDA	OUT
7	CTSA	IN
9	RTSA	OUT
13	GND	

Jb Pinout

Pin Number	Name	Direction
1	GND	
3	RXDB	IN
5	TXDB	OUT
7	CTSB	IN
9	RTSB	OUT
13	GND	

7. <u>Parts</u> <u>List</u>

PART NUMBER	DESCRIPTION	PER	LOCATION
1001008125	Data delay device, DDU-222-125	1	U10
1001100327	Oscillator, 32.768 Khz	1	XTAL1
1001100368	Oscillator, 3.68Mhz TTL Crystal	1	XTAL2
1001110001	Battery, 3V lith coin	1	B1
1001120001	Battery, holder coin	1	B1
1001210003	Switch, T101MH9AVB	1	SW1
1002001120	Socket, dip, machine, 20 pin	3	U8,U9,U19
1002001128	Socket, dip, machine, 28 pin	2	U12,U13
1002006810	Socket, pga, machine, 64 pin	3	U1,U3,U11
1003023226	Capacitor, tantalum, 22ufd	2	C3,C4
1003032100	Capacitor, ceramic, 10pfd, .100 rad	3	C1,C5,C6
1003032102	Capacitor, ceramic, 1000pfd, disk	1	C8
1003032334	Capacitor, ceramic, .33ufd, 256kramguard	14	C7,C9-C21
1003218001	Capacitor, trimmer, 2.1pf-18pf	1	C2
1004000101	Resistor, 100 ohm	1	R3
1004000103	Resistor, 10k ohm	4	R1,R7,R8,R10
1004000201	Resistor, 200 ohm	1	R6
1004000330	Resistor, 33 ohm	1	R2
1004000471	Resistor, 470 ohm	3	R4,R5,R11
1004000472	Resistor, 4.7k ohm	4	R9,R20,R30,R31
1004704729	Resistor network, Allen Bradley, 710A472	1	RP1
1005003904	Transistor, 2N3904	1	Q2
1005003905	Transistor, 2N3905	2	Q1,Q3
1005010746	Diode, 1N746	2	D1,D3
1005013600	Diode, 1N3600	1	D2
1006002505	Conn, 96 pin din, AMP # 532505-1	1	P1
1006011032	Conn, header standoff, 32 Pin BBS-132-GC	2.2	U2
1006032014	Conn, Molex 39-26-7148	2	J1,J2
1007020014	IC, 74LS14	1	U20
1007020245	IC, 74LS245	2	U6,U7
1007040373	IC, 74F373	2	U4,U5
1007168681	IC, MC68681	1	U16
1007201488	IC, 1488	1	U18
1007201489	IC, 1489	1	U17
1007204066	IC, 74HC4066	1	U14
1007258274	IC, MM58274	1	U15
1007420153	Pal, N82S153A	1	U19
1007421683	Pal,Tib, 16L8-15CN	2	U8,U9
1011006009	Front panel, 3U 8HP	1	
1011010361	PCA, SBCSACC, FAB 0320-0036 Rev A	1	

15

```
                    PARTNO   XXXXX ;
                    NAME     PAS  :
                    DATE     09/23/86 ;
                    REV      01 ;
                    DESIGNER Lehmann;
                    COMPANY  VME specialists ;
                    ASSEMBLY SBC3 ;
                    LOCATION U9   ;


/*******************************************************************/
/*                                                              */
/*                                                              */
/*                                                              */
/*******************************************************************/
/* Allowable Target Device Types:  PAL16L8-15                   */
/*******************************************************************/


/** Inputs **/

PIN      1  = !INBERR  :      /* BERR from system              */
PIN      2  = !MMUFLT  ;      /* MMU Fault                     */
PIN      3  = !CPSENSE ;      /* CO-proc is in place           */
PIN      4  = !CPCS    ;      /* CO-proc chip select           */
PIN      5  = !WR      ;      /* Write cycle                   */
PIN      6  = !AS      ;      /* 68000 AS/                     */
PIN      7  = !UDS     ;      /* 68000 UDS/                    */
PIN      8  = !LDS     ;      /* 68000 LDS/                    */
PIN      9  = !MAS     ;      /* MMU AS/                       */
PIN      11 = !DELAY   ;      /* Delay output                  */
PIN      14 = !WIN     ;      /* Write inhibit from MMU        */
PIN      15 = !RESET   ;      /* RESET/                        */


/** Outputs **/

PIN      12 = !PAS     ;      /* Physical address strobe to sys*/
PIN      13 = !DS      :      /* Data Strobe                   */
PIN      16 = !MMUCS   ;      /* Memory mangent chip select    */
PIN      17 = !PLDS    ;      /* LDS to system                 */
PIN      18 = !PUDS    ;      /* UDS to system                 */
PIN      19 = !PBERR   ;      /* Berr to system                */


/** Logic Equations **/

PBERR    =       INBERR
         #       MMUFLT
         #       CPCS & !CPSENSE ;


DS       =       PUDS
         #       PLDS ;


MMUCS.OE =       RESET :


MMUCS    =       'b'1;
```

1

```
FAS     =     AS & MAS & DELAY ;

FUDS    =     MAS & UDS & !WR
        #     MAS & UDS & !WIN ;

FLDS    =     MAS & LDS & !WR
        #     MAS & LDS & !WIN ;
```

```
          PARTNO   XXXXX ;
          NAME     IPL ;
          DATE     09/23/86 ;
          REV      .. :
          DESIGNER Lehmann ;
          COMPANY  VME Specialists
          ASSEMBLY S5C3 ;
          LOCATION U19  ;

/*********************************************************/
/*                                                      */
/*  Aux. board SIO 68681 autovector on level 1          */
/*                                                      */
/*********************************************************/
/*  Allowable Target Device Types: 82S153A              */
/*********************************************************/


/** Inputs **/

PIN     1  = !MEMCS  ;      /* Chip select for Static RAM  */
PIN     2  = !CLK    ;      /* Clock chip select           */
PIN     3  = !LDS    ;      /* 68000 LDS                   */
PIN     4  = !UDS    ;      /* 68000 UDS                   */
PIN     5  = !SIOINT ;      /* Interrupt from 68681        */
PIN     6  = !INIPL2 ;      /* IPL2 from system            */
PIN     7  = !INIPL1 ;      /* IPL1 from system            */
PIN     8  = !INIPL0 ;      /* IPL0 from system            */
PIN     9  = CLKCS   ;      /* Real time clock CS delayed  */
PIN     12 = !WR     ;      /* write cycle                 */

/** Outputs **/

PIN     11 = !RAMCLK  ;     /* ram OR clk selected          */
PIN     13 = !CLKWR   ;     /* Write cycle to real time clk */
PIN     14 = !MEMCSHI ;     /* Upper Mem CS                 */
PIN     15 = !MEMCSLO ;     /* Lower MEM CS                 */
PIN     16 = !DTACK   ;     /* DTACK for Memory, CLK        */
PIN     17 = !IPL0    ;     /* IPL0 to 68000                */
PIN     18 = !IPL1    ;     /* IPL1 to 68000                */
PIN     19 = !IPL2    ;     /* IPL2 to 68000                */

/** Logic Equations **/


DTACK.OE =   RAMCLK ;

DTACK    =   'B'1 ;

MEMCSHI =    UDS & MEMCS ;

MEMCSLO =    LDS & MEMCS ;

IPL2    =    INIPL2 ;
```

                              1

```
IPL1    =     INIPL1 ;

IPL0    =     INIPL0
        #     SIOINT & !INIPL1 & !INIPL2 ;

CLKWR   =     WR & LDS & !DTACK & CLK ;

RAMCLK  =     MEMCS
        #     CLKCS & CLK ;
```

```
                    PARTNO    XXXXX ;
                    NAME      CPA  :
                    DATE      09/23/86 ;
                    RE        01 :
                    DESIGNER  Lehmann :
                    COMPANY   VME specialists ;
                    ASSEMBLY  S&CT :
                    LOCATION  U8   ;

/**********************************************************************/
/*                                                               *.  */
/*                                                               */
/*                                                               */
/**********************************************************************/
/*  Allowable Target Device Types: PAL16L8A                      */
/**********************************************************************/


/**  Inputs  **/

PIN     1   = PA23     ;       /*  Physical address A23        */
PIN     2   = PA22     ;       /*                   A22        */
PIN     3   = PA21     :       /*                   A21        */
PIN     4   = PA20     ;       /*                   A20        */
PIN     5   = PA19     ;       /*                   A19        */
PIN     6   = PA18     ;       /*                   A18        */
PIN     7   = PA17     :       /*                   A17        */
PIN     8   = !DS      ;       /* Data Strobe                  */
PIN     9   = FC0      ;       /* 68000 Function code FC0      */
PIN     11  = !AS      ;       /* 68000 AS/                    */
PIN     14  = FC1      :       /* 68000 Function code FC1      */
PIN     15  = FC2      ;       /*                       FC2    */
PIN     18  = !PAS     ;       /* Physical address strobe      */


/**  Outputs  **/

PIN     12  = !CPCS    ;       /* 68881 chip select            */
PIN     13  = !SIOCS   ;       /* 68681 chip select            */
PIN     16  = !CLKCS   ;       /* Clock/calendar chip select   */
PIN     17  = !MMUCS   ;       /* 68451 chip select            */
PIN     19  = !MEMCS   ;       /* Static RAM/ROM chip select   */


/** Declarations and Intermediate Variable Definitions **/

PAD     = AS & PAS & DS & PA23 & PA22 & PA21 & PA20 & FC0 &
          !FC1 ;

/** Logic Equations **/

CPCS    = PAD &  PA17 &  PA18 & !PA19 ;

SIOCS   = PAD & !PA17 &  PA18 & !PA19 ;

CLKCS   = PAD &  PA17 & !PA18 & !PA19 ;
```

1

MMUOSICE = PAL & PHIT + PHIE & PHIF ;

MMLIS = 0 . ;

MEMII = PAL & PHIT & PHIE & PHIF ;

VME SPECIALISTS, INC.
550 BREWSTER AVENUE, SUITE 01
REDWOOD CITY, CALIFORNIA 94063
ACCESSORY BOARD FOR SBC2

VME750

DATE: OCTOBER 24, 1986   SHEET 2 OF 3

VME SPECIALISTS, INC.
550 BREWSTER AVENUE SUITE #1
REDWOOD CITY, CALIFORNIA 94063

ACCESSORY BOARD FOR SBC2

VME750

DATE: OCTOBER 24, 1986   SHEET 3 OF 3

Appendix N - Mach II Forth Manual

MACH 2

FORTH 83
Development System

Palo Alto Shipping Company

TABLE OF CONTENTS

i

TABLE OF CONTENTS

Appendices

General Information

Palo Alto Shipping Company	Sales Line:	(800) 44FORTH
P.O. Box 7430	Business Line:	(415) ~~054-7994~~ 363-
Menlo Park, CA 94026	Business Hours:	M-F, 11-4 (PST) 1399

Authors

Derrick Miley   Lori Chavez

Special Thanks

Terry Noyes, Aleksey Novicov, and Tim Lee.

Larry Leifer, Professor of Mechanical Engineering at Stanford University for creating
the Smart Product Design course and for introducing Mechanical Engineers to FORTH
year after year.   The Ed Wern School of Logic for the Wernian search algorithm.

Support

Technical questions should be directed to the Mach2 RoundTable on GEnie.
A Palo Alto Shipping representative will attempt to answer user questions on a daily basis.
We are limiting the usage of our business phone to introductory questions only.

GEnie

One-time registration fee: $18.00. Non-prime time access fee: $5.00/hour.
Order from: GENERAL ELECTRIC INFORMATION SERVICES COMPANY.
Voice: 1 (800) 638-9636. To join via modem:

1)   Set your modem for local echo (half-duplex) at either 300 or 1200 BAUD.
2)   Dial 1-800-638-8369 . When connected, enter   HHH
3)   At the U# prompt enter: XJM11912,GEnie
4)   Provide billing information and choose your password.

After credit confirmation, you will be able to use your password to enter the GEnie network.
Upon entering GEnie, type   MACH2   at any prompt to get to the Mach2 RoundTable.

Registration

If you did not purchase Mach2 directly from PASC, please the following information to PASC:

1)   Telephone number
2)   Street address
3)   Description of your hardware (computer type, amount of memory, disk drive capacity)
4)   Location where you purchased Mach2
5)   Date of purchase
6)   Mach2 version number.

Those who purchase Mach2 directly from PASC are automatically registered.

iii

Introduction

ABOUT THIS MANUAL

The primary goal of this manual is to give you the information you'll need to get the most out of this interactive development system. The information is presented in three different formats:

Discussion

The front section contains discussions on various MACH 2 programming topics. As you read through this section of the manual, use the glossary and appendices as sources of additional information about unfamiliar words or topics .

Glossary

The middle section is a glossary. The glossary is arranged alphabetically, most words are explained in a word/page format.

Appendices

The final section contains the appendices. The appendices contain very detailed information about the MACH 2 system and also tables of information which are too lengthy to be included in the front section.

OS-9/68000 Technical Documentation References

The revision letters for the OS-9/68000 technical documentation references in this manual are: Operating System Technical Manual - Revision F; Macro Assembler User's Manual - Revision D; Operating System User's Manual - Revision F; C Compiler User's Manual - Revision C.

---

m Beginning FORTH Programmers

We recommend that you use a beginner's FORTH-83 manual along with this manual. The glossary is a good source of FORTH programming examples. The demonstration programs included with MACH 2 are a good source of OS-9 (and MACH 2) programming examples.

b Experienced FORTH Programmers

Some new areas you might want to explore are local variables, the infix assembler, the symbolic disassembler, subroutine threading and macro substitution, and the MACH2-OS-9 interface.

PHILOSOPHY OF MACH 2

The major objective behind the development of MACH 2 has been to provide an advanced, interactive programming environment for software developers where the programmer's train of thought is not disturbed or limited by the programming environment itself.

FORTH is an appropriate language to fulfill this objective because, as a computer language, it encourages freedom of expression. However, this same freedom carries with it the responsibility of establishing self-imposed standards of communication with other programmers.

Standards:

Thus, one of our goals has been to adopt standard programming interfaces wherever possible. This has been done by using standard source files that can be edited by any text editor. This has also been done by providing a standard (infix) FORTH assembler that uses generic 68000 assembly syntax and incorporates as many of the functions found in the OS-9/68000 Macro Assembler as possible. Also included is a symbolic disassembler.

OS-9 Compatibility:

Another goal was to put the entire power of the OS-9 Operating System at the fingertips of MACH 2 users. This has been achieved by features such as high-level support of OS-9 user trap handler modules, interactive execution of OS-9 utility commands, single-step generation of compact, stand-alone turnkey applications, and OS-9 floating point support.

Speed:

One last goal has been to make MACH 2 run as fast as possible so that the programmer doesn't have to worry about programming for speed. The foundation of MACH 2 is a subroutine-threaded FORTH with automatic macro substitution which results in code that runs at a speed comparable to a compiled high level language-code which INHERENTLY runs 2-3 times faster than ANY other FORTH system. This gives the programmer the combined advantage of the powerful debugging features of an interactive language and the speed of a compiled language, at the same instant !

GETTING STARTED WITH MACH 2

With MACH 2 you should have received this manual, the MACH 2 master disk with the MACH2 application and additional MACH 2 demonstration programs.

The only item you need now to create your own software for the OS-9/68000 Operating System is a system with at least the minimum 256K of memory required by OS-9.

Starting MACH 2 Up

The MACH 2 application is named MACH2 and is located in the CMDS directory.
To enter MACH 2 type MACH2. After a few moments the "Palo Alto Shipping Company" prompt will appear. At this point, if you hit the carriage return key you should get an "ok". You are now ready to try the examples in the manual and on your distribution disk.

A QUICK INTRODUCTION TO FORTH

What follows is a brief introduction to the FORTH language. This is by no means a complete exposition of the language, but if you have never programmed in FORTH before it will give you an idea of what FORTH is and how easy it can be to program in FORTH. For a more thorough description of FORTH we suggest you purchase a beginner's manual on FORTH from your local computer bookstore.
(See the appendices for a list of suggested readings.)

Some Guidelines....

In the following examples the boldface type is used to indicate your inputs. The plain type is used for MACH 2's responses. Note that all spaces in your inputs are significant. FORTH expects all words and numbers to be separated from each other by spaces or tabs.The letter case (i.e. upper or lower) is not significant in MACH 2 (see the LOWER-CASE glossary page). A '<cr>' indicates that you should press the carriage return key. You may use the backspace key to correct any typing mistakes on the current line. If MACH 2 ever responds with a '?' or a '<name> ?' you have done something MACH 2 does not understand or like. Try retyping the line or changing the vocabulary search order (see the ONLY and ALSO glossary pages).

The MACH 2 Prompt

After you see the "Palo Alto Shipping Company" prompt try pressing your carriage return key several times. Every time you press a carriage return you will see the FORTH prompt

> **<cr>** ok <0>

The 'ok' means as far as FORTH is concerned, everything is going well. The '<0>' is the MACH 2 parameter stack (the parameter stack is the main stack in FORTH, often referred to as just 'the stack') depth indicator (see the BASE glossary page) .

FORTH is Extensible

The foundation of FORTH is the "FORTH dictionary". The FORTH dictionary is exactly that: a collection of words that define a language. Each word in the FORTH dictionary, when called upon, performs a specific action. This action may be as simple as moving a number from one part of computer memory to another, or as complex as writing data to a floppy disk. This collection of words is also referred to as the "FORTH kernel".

> **EXAMPLE 1: Executing a FORTH word.**
>
> We are now going to interactively execute a FORTH word.
> The word we will execute is the FORTH word 'WORDS'.
> This word will list the names of all of the words in the
> FORTH dictionary to the screen. To execute a FORTH
> word you type the name followed by a carriage return:

```
WORDS   <cr>
TURNKEY         MAKEMODULE     TCALL        DUMP
.S              -              ASCII        ASSIGNMODULE
$               QUIT           \            (
DEPTH <cr>
Hit the spacebar to continue. <cr>
ok <0>
```

Pressing any key will temporarily stop the listing. When the
listing is stopped you can press the space bar to continue
the listing or any other key to terminate the listing.

Now, writing a program in FORTH simply consists of defining a new word in terms of
words that already exist in the dictionary. Once a new word has been defined, it is
then added to the dictionary and can then be used in the definition of another word!

EXAMPLE 2: Defining new FORTH words.

Now we will create two new FORTH words. The names of our
FORTH words will be STAR and 3STAR:

```
: STAR   42 EMIT ;   <cr> ok <0>
: 3STAR    STAR STAR STAR ;   <cr> ok <0>
```

Now that the new words have been defined we may execute
them (remember, to execute a FORTH word you type the
name followed by a carriage return):

```
STAR   <cr>  * ok <0>
3STAR   <cr> *** ok <0>
```

We start the definition of a new word with a colon. The colon indicates to FORTH that
we are about to define a new word. The name which immediately follows a colon
(STAR in the above example) will be the name of the new word. A semi-colon marks the
end of the definition of a word. Any words between the name and the semi-colon
determine what the word will do when executed.

STAR's actions will be 42 EMIT. EMIT is a predefined word that prints out the character
which corresponds to the ascii value passed to it. The semicolon marks the end of the
definition of STAR. Since 42 is the ascii value for an asterisk, when STAR is executed,
an asterisk will be printed on the screen.

After STAR is defined, it becomes part of the FORTH dictionary and can be used in the definition of other FORTH words such as 3STAR. 3STAR's actions will be to print 3 consecutive asterisks to the screen. The ability of a computer language to extend itself in terms of itself is called extensibility. This is one of many desirable features that is an integral part of FORTH.

EXAMPLE :   Using the FORTH word 'WORDS' again.

We will now execute WORDS again to verify that our two new words, STAR and 3STAR have been appended to the FORTH dictionary. If they have been appended they should show up in the listing of all FORTH words:

WORDS <cr>

STAR	3STAR	TURNKEY	MAKEMODULE
TCALL	DUMP	.S	·
ASCII	ASSIGNMODULE	$	QUIT

\   <cr>
Hit the spacebar to continue. <cr>
ok <0>

FORTH is an Interpretive Language

If, in Example 2, you had only typed 42 EMIT, an asterisk would have been printed on the screen. You would have interactively executed the word EMIT. But what happened when you defined STAR in the above example? Why was no asterisk printed on the screen when you typed EMIT then? In FORTH there are 2 modes of operation:

(1) Compiling mode, which is invoked by a colon to
    indicate you are defining the actions of a new word.

(2) Interpreting or execution mode.

In FORTH, whenever you hit the carriage return and get an "ok" printed on the screen, you know you are in the execution mode. In this mode you can interactively execute any word, or sequence of words that exist in the dictionary. For example, try typing "3STAR STAR" followed by a carriage return and see what happens. This ability to execute any word in the dictionary immediately is a consequence of FORTH being an interpretive language. Interpretive refers to the fact FORTH interprets what you typed in and executes it immediately.

The power of an interpretive language is that you can quickly, and interactively, try out any sequence of words that you have defined. This provides an environment where you can test out your program at any level, modify it, and test it all out again in seconds! If you've programmed before you are probably familiar with the notion that an interpretive implementation of a language can be notoriously S L O W. However, MACH 2's internal structure is such that it runs at the same speed as traditional compiled languages such as C or PASCAL. With MACH 2 you get speed and an interactive development environment together. The following diagram illustrates this concept:

	BATCH	INTERACTIVE
COMPILED	<u>C</u> Faster Execution Slower Development	<u>FORTH</u> Faster Execution Faster Development
INTERPRETED		Slower Execution Faster Development <u>BASIC</u>

<u>Language Execution Method</u>

<u>Development Method</u>

FORTH has a Stack

Having a collection of words that perform certain actions is fine, but how do these words communicate or pass data? Before answering this question the concept of the FORTH parameter stack needs to be introduced.

In computer lingo, another name for the parameter stack might be called a LAST-IN-FIRST-OUT (or LIFO) queue. Think of the stack as an very tall tennis ball can that can hold a large number of tennis balls. Let's say you put in a tennis ball with #7 on it, and then put one in that has #2 on it, and then one with #4 on it.

If you go back to take one ball out, you are going to get the #4 ball. And if you prefer to use the #7 ball, you are going to have to take out the #4 and #2 tennis balls back out before you can take out the #7 tennis ball. This is exactly how the parameter stack works.

Last ball in is first ball out.

First ball in is last ball out.

8

Introduction to FORTH

Back to how FORTH words communicate by passing data or parameters. Some words need data on the stack to perform their specified actions. During execution these words will take data off of the parameter stack. Other words leave results on the stack after they have completed execution so that other words may use the results. Some words take parameters from the stack AND leave results on the stack while other words may not affect the contents of the parameter stack at all. To aid the programmer, all FORTH words listed in the glossary use 'Stack Notation' to indicate how the particular word will affect the contents of the parameter stack. The concept of 'stack notation' is explained in the FORTH Topics section.

EXAMPLE 4: Placing numbers on the FORTH parameter stack.

To put a number on the FORTH parameter stack you simply type the number (or several numbers separated by spaces) followed by a carriage return. The stack depth indicator, the '<n>' which follows the 'ok' prompt will always tell you how many numbers are currently on the parameter stack:

```
35 <cr> ok <1>        ( put 1 number on the stack )
3 -4 <cr> ok <3>      ( put 2 more numbers on the stack )
                      ( there are now 3 numbers on the stack )
```

The stack depth indicator is also an indicator of what number base is currently being used for all numeric I/O. A '$' in the stack depth indicator indicates that the current base is hexadecimal:

```
HEX <cr> ok <$3>      ( change the number base to hexadecimal )
E <cr> ok <$4>        ( put another number on the stack )
```

EXAMPLE 5: Displaying the numbers on the stack.

One way to display a number on the stack is to use the FORTH word '.' ("dot"). '.' will take the top number off the stack and display it:

```
. <cr> E ok <$3>      ( take the top number off the stack and display it )
```

The FORTH word '.S' ("dot-S") will produce a non-destructive display of the numbers on the stack (i.e. you don't have to take the numbers off the stack to see them):

```
.S <cr>
23 3 -4  <- Top
  ok <$3>
```

Notice that the stack depth indicator shows that our three numbers are still on the stack. Since we are in hexadecimal base our decimal 35 is displayed as a hexadecimal 23.

9

Introduction to FORTH

An example of a FORTH word that takes data off of the stack and also leaves a result
on the stack is the math operator "+". This FORTH word, called "plus", takes 2 numbers
from the stack, adds them, and then puts the result on the stack:

EXAMPLE 6:   Stack arithmetic.

34 66 + . <cr> 100 ok <0>   (Put 2 numbers on the stack, add them
                                           together and then display the result)

3 50 70 - * . <cr> -60 ok <0>

The second example is a little more complicated. First we put
three numbers on the stack then, going left to right, we replaced
the top two numbers on the stack with a subtraction result
(50 - 70 = -20) and then replaced the remaining two numbers
(a 3 and a -20 at this point) with the result of a multiplication.
Then the result, a -60, was taken from the stack and displayed.

This method of arithmetic calculation is Reverse Polish Notation (or RPN) and is
commonly found on HP calculators.

A Special Note About the MACH 2 Stack

MACH 2 is based upon the FORTH-83 standard and the stacks used in MACH 2
are 32-bits wide. Most FORTH's, both 79 and 83 versions are 16-bit FORTH's. That is,
their stacks are 16-bits wide and the word @ (see the @ glossary entry) would return
only 16-bits of data. In MACH 2 a @ returns 32-bits of data, a W@ returns 16-bits of
data, and a C@ returns 8-bits of data.

FORTH is a Structured Language

In any kind of a program, different actions are taken depending on external input,
whether this input comes from a user at the keyboard, or from a photodiode sensor
attached to an input port of the computer. In order to determine which actions to take,
decisions have to be made.

In FORTH there exist standard control structures that control the flow of a program
depending on the decision made. These control structures have the same inherent
structure at any level of a program. This is the basis of a structured language.

EXAMPLE 7:   Using the IF...ELSE...THEN program control structure.

In the following example a word is defined that looks to see if a
number on the stack is a 1 or a 0. IF it is a 1, it prints an asterisk
(42 EMIT), ELSE it prints a plus sign (43 EMIT):

10

Introduction to FORTH

```
: PLOT ( n - )
   IF
         42  EMIT
   ELSE
         43  EMIT
   THEN   ;

1  PLOT  <cr>  * ok <0>
0  PLOT  <cr>  + ok <0>
```

There are only 8 basic control structures that you will ever need when writing any software. These control structures are the following:

1)   IF...THEN
2)   IF...ELSE...THEN
3)   BEGIN...UNTIL
4)   BEGIN...WHILE...REPEAT
5)   BEGIN...AGAIN
6)   DO...LOOP
7)   DO...+LOOP
8)   CASE...OF..ENDOF...ENDCASE

More information about these program control structures may be found on the corresponding pages in the glossary section.

Introduction to FORTH

A QUICK INTRODUCTION TO MACH 2

Now that you have mastered the basics of FORTH you are ready to take a quick tour
of MACH 2's features. This is not intended to be a complete MACH 2 tutorial but rather
a demonstration of the use of the major MACH 2 features. Refer to the table of contents
or the index to find a more complete description of any particular feature.

Checking Your Memory Availability

There are three memory areas which are important to MACH 2 and your program: the
code space (where your program code will be located), the names space (where the
FORTH headers for your code will be located), and the variable space (where your
program variables will be located). The FORTH word '?FREE' will tell you how much
space is currently available in each of these five areas:

        ?FREE <cr>
        Code : 32768
        Vars : 13390
        Name : 16384
        ok <0>

The results of ?FREE are system dependent, the numbers you see will probably be
different than those above.

Using Local Variables

In the next definition we will make use of MACH 2's local variables. Local variables help
eliminate confusing stack manipulations by allowing you to assign names to values on
the stack. Later, when you need the value, you execute its name to put the value on
top of the stack. The word #TIMES will keep a count of how many times a BEGIN...UNTIL
loop was executed before the loop was terminated by a keypress. #TIMES will keep
the running count in a local variable:

```
: #TIMES {   |  total  -    }
        0  ->  total                \ zeroing the local variable
        BEGIN                       \ start of the loop
                1  +>  total        \ increment the count each time through the loop
        ?TERMINAL                   \ has a key been pressed ?
        UNTIL
        total  .  ;  <cr>ok <0>     \ display the total
```

12

Introduction to MACH 2

Local variables and named input parameters (this definition does not use named input parameters) are defined between the left and right curly brackets. #TIMES uses one local variable named 'total' during its execution. Execute #TIMES to see how it works:

```
#TIMES  <cr> ok <0>
( wait a bit... )
<cr> 2212  ok <0>          \  pressing any key will stop execution of #TIMES
                           \  the number of loops executed will be displayed
```

Executing OS-9 Utility Commands

The MACH2 '$' command allows you to interactively execute OS-9 utility commands from within MACH2:

```
$  DIR
            Directory of .  01:14:11
CMDS        OS9Boot       SYS         key.a
read        sieve         startup

ok <0>
```

Since MACH2 is a re-entrant module, you can use '$' to temporarily suspend this version of MACH2 and enter another version:

```
$  MACH2   <cr>           \ Use '$' to re-enter MACH2
Palo Alto Shipping Company \ Now we're in...

BYE  <cr>                 \ Use BYE to exit the second version.
  ok <0>                  \ ...and now we've returned to the original MACH2
```

Note that this second version of MACH2 starts up much faster than the original version since the MACH2 module has already been loaded into memory.

Creating a TURNKEY Executable Module

In MACH2 it is very simple to create stand-alone, executable OS-9 modules.
The following example shows how TURNKEY is used to create an executable module:

```
: NumberOne  (  -  )
     5  0  DO
              ."  This is my first MACH2/OS-9 executable module !!   "
              CR
      LOOP  BYE  ;  <cr> ok <0>

TURNKEY  NumberOne  First  <cr>
```

At this point MACH2 will have returned you to the OS-9 shell. Type 'First' to execute
your module (after your module has completed execution, re-enter MACH2):

```
$  First  <cr>
This is my first MACH2/OS-9 executable module !!
This is my first MACH2/OS-9 executable module !!
This is my first MACH2/OS-9 executable module !!
This is my first MACH2/OS-9 executable module !!
This is my first MACH2/OS-9 executable module !!
$
```

Floating Point Calculations

MACH 2 lets you use the floating point routines provided by Microware for OS-9.
As an example of their use, let's create a FORTH word which calculates the area of the
circle whose radius is passed to it on the floating point stack. We will use the formula:
area = πR^2 :

```
ALSO  MATH  <cr> ok <0>
FP  <cr> ok <0> [0]                  ( Change to floating point input mode.
                                       Note the addition of a floating point
                                       stack depth indicator. )
: CircleArea  (  fp-radius  -  fp-area  )
      FDUP                           ( Duplicate the input )
      F*                             ( Square the input )
      3.14  F*  ;  <cr> ok <0> [0]   ( Multiply by $\pi$ )

4.25  CircleArea  <cr> ok <0> [1]    ( Ask for area of circle with 4.25 radius )
F.  <cr> 56.7163  ok <0> [0]         ( Display the floating point result )

INT  <cr> ok <0>                     ( Switch back to integer input mode )
```

14

Introduction to MACH 2

Loading and Executing Programs

The MACH2 word ' INCLUDE" ' is used to load text program source files. INCLUDE" is followed by one space, the name of the file to be loaded, and a trailing ' " ' (the trailing quote should immediately follow the file name). The 'sieve' program is on your distribution disk:

    INCLUDE" sieve" <cr>

    Type 'sieve' to execute this benchmark program
     ok <0>

    sieve <cr>

    4 Secs 54 Ticks | 128 Hertz
    1899 primes ok <0>

EMPTY

To reclaim all of your code space, variable space, and names space use 'EMPTY'. EMPTY will remove all of your new definitions and variables from the dictionary.

    EMPTY <cr>
     ok <0>

Using the Assembler

The MACH 2 assembler is always available. In this example we will create an assembly language definition (a 'code' definition) which will add 1 to the number on top of the stack, if the number is odd, when executed:

```
CODE  IncrOdd  ( n1 -- n2 )
        MOVE.L  (A5),D0      \ Move number into data register
        BTST    #0,D0        \ Check for odd number
        BEQ.S   @1           \ Exit if number is even
        ADDQ.L  #1,(A5)      \ Add 1 to odd number
@1      RTS                  \ End of routine
END-CODE   <cr> ok <0>
```

The important thing to notice in this example is that MACH 2's assembler is a standard, infix assembler. It isn't the simplified RPN assembler commonly included in other FORTH implementations. We think this makes assembly routines written in MACH 2 much more readable and supportable. Let's try 'IncrOdd':

    4 IncrOdd . <cr>
     ok <0>
    9 IncrOdd . <cr>
     ok <0>

15

Introduction to MACH 2

Using the Symbolic Disassembler

The symbolic disassembler allows you to look at how your definitions were compiled into memory:

```
'    IncrOdd  7  IL  <cr> ok <0>

IncrOdd
055232:  MOVE.L    (A5),D0
055234:  BTST      #$0,D0
055238:  BEQ.S     +$4            ;  $5523C
05523A:  ADDQ.L    #$1,(A5)                         1+
05523C:  RTS
 ok <0>
```

Using the MACH2 Debugger

The MACH2 debugger allows you to watch your code in action. Because OS-9 is a multi-tasking operating system, only one person may be using the MACH2 module when the MACH2 debugger is being used (read the debugger section for more information). The following example shows how our 'IncrOdd' word could be debugged using the MACH2 debugger (you enter the boldface commands):

```
DEBUG    IncrOdd    <cr>   ok <0>
5    IncrOdd    <cr>

055232:    MOVE.L    (A5),D0                              \ this is the first instruction
PC:00055232  SR:0310              X=1 N=0 Z=0 V=0 C=0      \ in IncrOdd
A0:00051154  A4: 00000000  D0: 00055232  D4: 00000008
A1:00051152  A5: 0001F1E6  D1: 00000007  D5: 00000007
A2:00000000  A6: 00021100  D2: 00000003  D6: 00000007
A3:0001EA32  A7: 0002017E  D3: 00000000  D5: 0001EA0A


> S   <cr>                                                \ 'S' means single-step
055234    BTST      #$0,D0
PC:00055234  SR:0310              X=1 N=0 Z=0 V=0 C=0
A0:00051154  A4: 00000000  D0: 00000005  D4: 00000008
A1:00051152  A5: 0001F1E6  D1: 00000007  D5: 00000007
A2:00000000  A6: 00021100  D2: 00000003  D6: 00000007
A3:0001EA32  A7: 0002017E  D3: 00000000  D5: 0001EA0A


> SS   <cr> 5    <Top                                     \ show the contents of
                                                          \ the FORTH param stack

> G   <cr>                                                \ Go ahead with the
 ok <1>                                                   \ rest of the word.

.    <cr>6  ok <0>
```

16

Introduction to MACH 2

MACH 2 Forth Topics

MACH 2 IN MEMORY

There are three memory areas of interest in the MACH 2 system. These are the code, names, and variable areas used by the programmer.

CODE SPACE

The code space (dictionary space) is the place where the code portion of your definitions is placed. MACH 2 keeps its definition headers separate from the definition code. The headers (or names) for a definition are placed in the names space (discussed later). Each time a definition is added, the pointer to the next available code space location, the HERE pointer, is incremented accordingly. The following diagram shows how the addition of the definition TEST affects the HERE pointer:

: TEST DUP SWAP ROT ; <cr> ok <0>

```
                           ◄──────────────────── Top address in
Higher Memory   ┌────────┐                        your code space.
                │        │
                │Remaining│
                │Code Space│
                │        │
                │   ⬆    │
                │        │
                ├────────┤   ◄── top of dictionary  ◄── HERE
                │Code Space│      after 100 ALLOT
                │reserved by ALLOT│
                ├────────┤   ◄── top of dictionary
                │JSR ROT │      after TEST
                │code for SWAP│
                │code for DUP│
                ├────────┤   ◄── top of dictionary
                │▒▒▒▒▒▒▒▒│      before TEST
                │▒▒▒▒▒▒▒▒│
                │▒▒▒▒▒▒▒▒│   ◄── previous definitions
Lower Memory    │▒▒▒▒▒▒▒▒│
                └────────┘   ◄──────────────────── Base address of
                                                  your code space.
        Code (Dictionary) Space
```

When you first start up in MACH 2, the HERE pointer will be pointing at the base address of you code space. Each time you add a definition, the HERE pointer is incremented so that it is always left pointing at the next available spot in the code space. ALLOT is a word you can use to artificially increment the HERE pointer.

Using ALLOT to Allocate Dictionary Space

The word ALLOT may be used to explicitly increment the HERE pointer by a specified number of bytes. ALLOT is commonly used with CREATE to reserve space in the dictionary for tables of static data such as trigonometric tables:

```
CREATE  Sines  <cr> ok <0>    ( CREATE a dictionary header for Sines )
100  ALLOT  <cr> ok <0>       ( Reserve 100 bytes in the code space
                                for the sine table data. )
```

In this example, the defining word CREATE is used to create the dictionary header for a table of sine information. When the word SINES is later executed it will push the address of the first byte of the sines table on the stack. This allows the data in the table to be accessed. Since the headers are stored in a different location, the execution of CREATE did not affect the above diagram. ALLOT is used to allocate the necessary amount of space in the dictionary for the sine data. In the diagram, it can be seen that ALLOT moved the HERE pointer up by 100 bytes. These bytes have not been initialized to any value. Now the data may be stored in the table without fear of it being overwritten when the next definition is added.

To find the address of the next available spot in the code space execute the word HERE:

```
HERE  .  <cr> 54102 ok <$0>
```

Size of the Code Space

Unless otherwise specified, MACH2 will ask OS-9 for 32K of memory for the user code space upon start up. If you wish to override the default code space size pass your desired code space size to MACH2 when you start MACH2 up. For example, to set the user code space to size to 16K bytes:

```
$  MACH2    -$8000
```

The names space is always set to 1/2 the size of the code space.

THE NAMES SPACE

The names space is where the header portion of a definition is stored. The header for a definition contains information such as the definition name, the length of the definition name, a pointer to the next definition (each time a definition is added in a FORTH system it is linked into a linked list of all previously defined definitions), and a pointer to the actual code for the definition (located in the code space). The exact format of a MACH 2 dictionary header is given in the appendix.

MACH 2 keeps the headers separate from the code so that they may be discarded by TURNKEY when it creates your final application. The header information may be discarded since it is only used during compilation; header information is not required for program execution and would only take up valuable disk space if it were not discarded.

The diagram below shows the names space which corresponds to the code space shown in the previous diagram:

Higher Memory

Remaining
Names Space

← Top address in
your names space.

← top of names space
after TEST ← NP

← top of names space
before TEST

← previous headers

Lower Memory

← Base address of
your names space.

Names Space

A pointer called the NP (name pointer) is used to keep track of the remaining names space. Each time a definition is added, the new header information is placed into the names space starting at the address pointed to by the NP. The NP always points to the next available spot in the names space. After the header is in place, the NP will be incremented accordingly. To get the address of the next available spot in the names space you can execute the word NP (to put the address of the NP system variable on the stack) and then 'fetch' its contents:

        NP @ . <cr> 201AE ok <$0>

Size of the Names Space

When you start up in MACH 2 you are given an empty 16K space for your program names. You can use ?FREE at any time to check the amount of room left in your current names space.

20

MACH 2 in Memory

THE VARIABLE SPACE

The variable space is where a program's variable data (data which will be altered during the running of the program) should be stored. Since MACH 2 keeps the variable and dictionary space separate, the only information about the variable space which needs to be saved on disk as part of a program is the size of the variable space.

Location of the Variable Space

The variable space is located relative to the address contained in the A6 register of the CPU. The data area used for the variable space is dynamically allocated by OS-9 when a module is loaded. As the diagram below shows, MACH2 asks OS-9 for a 16K data area. 4K of the data area is used for MACH2 jump tables. The remaining 12K is used for the variable area. The address in the A6 register is always located 32K bytes above the base address of the data area.

21

MACH 2 in Memory

Size of the Variable Space

Initially, you will be given 12K bytes of variable space. To adjust the size of the variable space use the 'MACHVarSpace' utility found in the 'CMDS' directory. Do not use this utility from within MACH2 ! This utility will modify MACH2's module header information to accommodate your new desired variable space size. The next time you enter MACH 2 your requested amount of variable space will be set aside. Note that there is a 64K limit on the size of a module's "local" data area. If you request a variable space larger than 60K (4K is always taken by Mach2) the compiler will use longer addressing modes to access data in the remote data areas. ?FREE may be used to check the amount of free variable space remaining.

Creating New Variables

The word VARIABLE is used to create new, named variables. VARIABLE will create a dictionary entry for the variable using the specified name.

        VARIABLE Array <cr>ok <$0>

To get the address of the variable location in the variable space you execute the name of the variable:

        Array . <cr>67804 ok <$0>

When VARIABLE creates the dictionary entry for the variable the dictionary header goes in the names space, the code responsible for calculating the address of the variable when its name is executed goes in the code space, and 4 bytes of storage space for the new variable are automatically reserved in the variable space. Initializing the contents of these 4 storage bytes is the responsibility of the programmer.

Storing and Retrieving Data To and From Variables

The word '!' ('store') is used to place 4 bytes of data at a specific address and the word '@' ('fetch') is used to retrieve 4 bytes of data from a specified address.

        3F Array ! <cr> ok <$0>
        Array @ . <cr> 3F <$0>

'W!' , 'W@' , 'C!' , and 'C@' are used to store and retrieve 2 and 1 byte lengths of data.

Using VALLOT to Allocate Variable Space

If you wish to create a variable which has more than 4 bytes of storage (if you want to store an array of data for example) use VALLOT immediately after you have created the variable with VARIABLE:

        VARIABLE Array <cr> ok <$0>   (Create the variable)
        400 VALLOT <cr>ok <$0>   (Allocate 400 extra storage bytes
                                                   for the contents of the previously
                                                   defined variable.)

22

MACH 2 in Memory

A 'pointer' called the VP is used to mark the next available spot in the variable space. Note that the VP works VERY differently than the HERE pointer or the NP. VP is the name of a MACH 2 system variable which holds a negative offset from the address in the A6 register to the next available spot in the variable space.

      VP @ . <cr> -35B4 ok <$0>

VALLOT will decrement the VP pointer by a specified number of bytes. This has the effect of reserving storage locations in the variable space.

?FREE - A MACH 2 MEMORY UTILITY WORD

?FREE is a utility word which tells you the amount of free space currently available in your code, names, and variable areas.

      ?FREE <cr>

      Code : 32768
      Vars : 10000
      Name : 16372
      ok <0>

The sizes of certain memory areas are system dependent so the numbers ?FREE returns on your system will probably be quite different than those returned in the above example.

VOCABULARIES

In FORTH, a vocabulary is a linked list of definitions. Vocabularies are used to separate the definitions into meaningful groupings. The FORTH dictionary is comprised of several vocabularies.

The MACH 2 Vocabularies

The initial MACH 2 dictionary consists of 4 vocabularies. These vocabularies are :

        FORTH   :All of the standard (FORTH-83 and others) words.
        OS-9   :All OS-9-specific words.
        ASSEMBLER   :All words used with the Assembler/ Disassembler.
        MATH   :All of the floating point words.

Specifying a Search Order

When a FORTH word is executed immediately (by typing its name at the keyboard) or when it is compiled (by using it within a colon definition) the dictionary must be searched to find out what the word should do. Since the dictionary is broken up into at least four vocabularies (the four mentioned above), MACH 2 must be told which vocabularies it should search through and in which order it should search. The two FORTH words ONLY and ALSO are used to specify a search order for MACH 2.

ONLY

ONLY, as might be expected, is used to specify that ONLY one vocabulary should be searched when MACH 2 is looking up a word. The name of that one vocabulary should follow ONLY :

        ONLY   FORTH   <cr> ok <0>

The use of ONLY above tells MACH 2 that it should only search through the FORTH vocabulary's list of words. If the word hasn't been found by the time the end of the vocabulary list is reached, MACH 2 will display a '<name> ?' error message.

ALSO

ALSO is used to specify that another vocabulary should ALSO be searched by MACH 2:

        ALSO OS-9   <cr> ok <0>

If this were used after ONLY FORTH above, MACH 2 would know that it should search through the OS-9 vocabulary list first, and then, if it reached the bottom of the OS-9 list without finding the word, it should start searching through the FORTH vocabulary list. ALSO may be used to specify up to 5 additional vocabularies to be searched. Seven vocabularies may be included in the search order at once. ALSO <name> will cause the <name> vocabulary to be searched first.

Adding Definitions to a Vocabulary

The word DEFINITIONS is used to specify to which vocabulary list new definitions should be appended :

<div align="center">

DEFINITIONS   <cr> ok <0>

</div>

DEFINITIONS will make the vocabulary which is currently being searched first (the 'transient' vocabulary) the vocabulary to which all subsequent definitions will be added. In this case, since OS-9 has just been made the transient vocabulary by ALSO, the OS-9 vocabulary would now be the vocabulary to which definitions will be appended.

After,
ONLY   FORTH   <cr> ok <0>
ALSO   OS-9   <cr> ok <0>

the search order would be ....

New
Definitions

FORGET gets rid of
everything defined
after a certain point
in time, no matter
which vocabulary
the words are in.

EMPTY gets rid of
everything above
this protecting line.

OS-9
Vocabulary

MATH
Vocabulary

ASSEMBLER
Vocabulary

When the search order
is SEALed, the link to
the words used to modify
the search order is
snipped. From that
point on the application
may only use words
found in the 'frozen'
search order.

Protected
Words

FORTH
Vocabulary

Words used to modify
the search order

25

Vocabularies

ORDER and WORDS -- Two Search Order Utilities

The word ORDER will print out information about the current search order. It will show
which vocabularies are being searched, in which order they are being searched, and
to which vocabulary definitions are currently being appended. Upon system start-up
the vocabulary structure will be set as follows:

> **ORDER** <cr>
> Search Order --> FORTH
> New Definitions --> FORTH
>   ok <0>

The word WORDS will display a listing of all words in the transient vocabulary.

Application Vocabularies

Certain applications may require that a limited set of application words are available
for execution from within the running application. This type of application may NOT
be sold or distributed unless special arrangements have been made with the
Palo Alto Shipping Company in advance.

The word SEAL is available for this purpose. SEAL will freeze the current search order
and will also remove the user's ability to change the search order by cutting the link
to the set of search order modification words. As the diagram on the previous page
indicates, the set of search order modification words is usually searched at the end of
every dictionary search.

Creating Vocabularies

New vocabularies may be created with the use of the defining word VOCABULARY.
MACH 2 may have up to 15 total vocabularies. Since four vocabularies already exist,
users may define up to 11 of their own. VOCABULARY is used as follows :

> **VOCABULARY MINE** <cr> ok <0>

Definitions may be added to the new vocabulary by using the words ONLY, ALSO,
and DEFINITIONS as described above.

Removing Words from the Dictionary

The word FORGET may be used to remove a definition, and all definitions added
after that definition, from the dictionary :

> **FORGET <definition-name>** <cr> ok <0>

The word EMPTY will remove all definitions which have been added to the dictionary
See the previous diagram for an illustration of the effect of these two words.

LOCAL VARIABLES AND NAMED INPUT PARAMETERS

Local variables are those whose contents are valid only inside the definition in which they are used. Because they are local variables, as opposed to global variables, they will support re-entrant code when writing recursive or multi-tasking programs. Local variables may have the same names in different definitions without conflicting.

Named input parameters, which are simply initialized local variables, are used to give names to the input parameters on the stack. This greatly simplifies routines which would otherwise require complex stack manipulation.

The following comparison shows that the use of local variables and named input parameters also greatly increases the readability of FORTH words which perform stack manipulations.

Contrasting Local Variables with Conventional FORTH Stack Manipulation

QUADRATIC EQUATION

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Local Variables

```
: Quadratic { a b c | radical 2A -- x1   ( '{ )' and '}' are used to set up local variable list )
        b b *
        4 a * c * -                   ( b² - 4ac )
        SQRT -> radical               ( take square root of number under radical )

        a 2* -> 2A                    ( 2a )

        b NEGATE radical - 2A /       ( calculate the + and - results ... )
        b NEGATE radical + 2A / ;     ( and leave both on stack )
```

Conventional FORTH Stack Manipulation

```
: Quadratic ( a b c - x1 x2 )         ( Parentheses are used for comments )
        2 PICK * 4 *                  ( 4ac )
        OVER DUP * SWAP -             (     b² - 4ac )
        SQRT                          ( take square root of number under radical )
        SWAP NEGATE 2DUP +            ( calculate the + root )
        3 PICK 2* / >R                ( divide by 2a and save on loop stack )
        SWAP - SWAP 2* /              ( calculate the - root, leave on stack )
        R> ;                          ( retrieve + root and leave on stack )
```

Both of these definitions will return the roots on the stack, however the example which uses named input parameters and local variables is MUCH easier to read and to write.

27

Local Variables

Specifying Local Variables In the Local Variable List

The local variable list is contained within the "{" and the "}".   In this example there are
four local variables, with two of them, "x" and "y", specified as named input parameters.
Any names before the "|" are considered named input parameters and require
corresponding data on the stack prior to execution of the definition. Any names after the
"|" and before the "-" are treated as local variables. The contents of local variables
are undefined until written to by using the '->' operator. Information after the '-' and
before the '}' is a comment used for stack notation.

```
  : SKEWED  { x y  |  x1 y1  -  result }
        x           ->  x1
        y           +>  x1
        x y -       ->  y1
        x1  y1  * ;
```

Named Input Parameters
are initialized from values
on the parameter stack.

```
3  4   :  HYPOT {     x   y   -   result }
          x x *
          y y *  +  ;
```

The number on top of the stack will
go to the rightmost named input
parameter in the list.

The '->' , '^' , and '+>' Operators

In the above example, the '->' (pronounced "save-to") operator is used to store the number
on top of the parameter stack, into the specified local variable. The '^' (pronounced "hat")
operator is used to obtain the address of a local variable (see the ^ glossary page for an
example of its use. The '+>' (pronounced "plus-to" operator is used to add the number on
top of the parameter stack to the specified local variable.

Speed Considerations

The set-up and initialization of local variables occurs at execution time. This does not
mean, however, that a word which uses local variables will take longer to execute. For
example, the local variable square root example on the previous page executes just as fast
as the non-local variable example. In most cases, the decrease in execution time due to
program simplification gained by the use of local variables will completely offset the
extra execution time required to set the local variables up.

Recursive Definitions Using Named Input Parameters

Local variables are especially well-suited to recursive definitions (see the Fibonacci example
on the demonstration disk). Since the local variables are implemented with a method called
stack framing, the amount of recursion that can occur in a recursive definition is limited only
by the available subroutine stack space.

28

Local Variables

STACK NOTATION

Stack notation is a very critical component of FORTH programs. Stack notation is used to describe how execution of a FORTH word affects the contents of the parameter stack. Here is an example :

```
: @CHAR   ( a n - c )   + C@  ;
```

In the above example, the stack notation corresponding to the definition @CHAR is in bold typeface. The letters to the left of the '-' specify the input parameters @CHAR expects to find on the parameter stack. The letters to the right of the '-' specify the output parameters @CHAR will return on the stack. By including this special FORTH comment immediately after each new word, the behavior of a FORTH word can be assertained at a glance.

General Format of Stack Notation

The following examples show the general format used for stack notation :

```
(    INPUTS    -    OUTPUTS    )
```

```
EX:  ( n1 n2 - n3 )    ( c )    ( - a )    ( n1 n2 - f )
```

If a word does not affect the contents of the parameter stack, stack notation may be omitted. If a word has only an input and no output, the dash is sometimes omitted.

Characters Used In Stack Notation

Four different lowercase characters are commonly used inside of a stack notation comment :

```
a  -  address
n  -  number
c  -  ascii character
f  -  boolean flag
```

Since MACH 2 is a 32-bit implementation of FORTH (i.e. its stacks are 32-bits wide), all addresses (a), numbers (n), characters (c), and flags (f) will be 32-bit values while on the stack.

Stack Notation with Named Input Parameters

When using Named Input Parameters, the local variable list should be used to indicate the stack notation for the word. The general format used for a local variable list is:

```
{    INPUTS    |    LOCAL    --    OUTPUTS    }
```

```
EX:  { x y | -- result }    { x y | }    { x y }    { | u v - double }    { | u v }
```

The OUTPUTS field, which follows the dashes, is treated as a comment and does not affect execution of the word. The OUTPUTS field in a local variable list should be used to indicate which values, if any, the word leaves on the parameter stack.

FLOATING POINT

MACH 2's floating point operators use the MATH module provided by Microware. This module contains routines for basic floating point math, extended integer math, type conversion, and transcendental and extended mathematical functions. All of the floating point words and operators that make up the MACH 2 floating point package are found in the MATH vocabulary.

Precision

The routines in the math module support the following data formats:

Integer types:	unsigned	32-bit unsigned integers
	long	32-bit signed integers
floating point:	float	32-bit floating point numbers
	double	64-bit double precision floating point numbers

The floating point math routines use formats based on the proposed IEEE standard for compatibility with floating point math hardware. 32-bit floating point operands are internally converted to 64-bit double precision before computation and converted back to 32 bits after as required by the IEEE and C language standards. (Therefore, the float type has no speed advantage over the double type.)

The precision of the following transcendental and extended math routines is set using the MACH2 word PRECISION. The precision may be set from 1 bit (1E-001) to 14 bits (IE-014) by passing a number from 1 - 14 to PRECISION.

```
FCOS      FSQRT
FATAN     FSIN
Flog      FTAN
Fln       Fy^x
```

The Floating Point Stack

An independent stack, called the floating point (FP) stack, is used for all floating point operations. This stack differs slightly from the parameter stack in that it has room for a maximum of 20 FP numbers only. Numbers on the FP stack can be manipulated with standard stack operators such as FROT, FOVER, FSWAP, FDROP, FDUP, etc. Floating point numbers can also be converted to integer and transferred to the parameter stack with the F>I operator. The I>F operator performs the reverse function.

Floating Point Mode

For floating point numbers to be recognized as such, you have to be in the FP mode. The FP mode is invoked by executing "FP" from the MATH vocabulary. Once you are in the FP mode, any numbers that are entered with a decimal or an exponent are put on the floating point stack. Examples of valid floating point numbers are shown below:

```
-1.65e-05          9119.
0.0045             345.88E+67
```

Note that in the FP mode any integers (numbers with no decimal point) that are entered are still put on the parameter stack. While in the FP mode you will notice a number in square brackets after every "ok". This is the floating point stack depth indicator and works the same way as the parameter stack depth indicator.

The FP mode is exited by executing "INT". This puts MACH 2 back in the integer mode and any subsequent FP numbers will not be recognized and the floating point stack depth indicator will not be shown.

Displaying Floating Point Numbers

Floating point numbers can be displayed with the "F." operator in a FIXED point format. To display numbers in the fixed point format, execute the word 'FIXED' preceded by the number of digits to follow the decimal point.

EXAMPLE:

```
ALSO  MATH  <cr>  ok <0>
FP  <cr>  ok <0> [0]
23.45678   <cr>  ok <0> [1]
FDUP  <cr> ok <0> [2]
2  FIXED  <cr>  ok <0> [2]
F.  23.47  <cr>  ok <0> [1]
NT  <cr> . ok <0>
```

Floating point numbers and operators may also be used within colon definitions:

```
ALSO  MATH  <cr> ok <0>
FP  <cr>  ok <0> [0]
: TEST  3.4  2.6  F°  ;  <cr>  ok <0> [0]
TEST  F.  <cr> 8.84000  ok <0> [0]
INT  <cr> ok <0>
```

For a summary of all of the floating point words and operators please refer to the floating point Glossary section. For more information on the OS-9 math module see the Math Module chapter in the OS-9/68000 Operating System Technical Manual.

TURNKEY

Traditionally, one of the hardest tasks for a FORTH system (or any interactive system) was the creation of stand-alone applications. Since the primary purpose of any development system IS the creation of applications, our goal was to make the MACH2 TURNKEY process as simple and as powerful as possible.

The MACH2 TURNKEY process is simple because it allows you to turn your programs into OS-9 executable program modules with just one line of code. It is powerful because it strips away the development environment code which is not required for a stand-alone application in order to create a minimal sized program module.

Using TURNKEY

Before using TURNKEY you must load your program into MACH2. When the loading process is complete, use TURNKEY as follows:

> **TURNKEY   <main-word>   <module-name>   <cr>**

<main-word> should be the name of the word in your program which performs all program initialization and starts the program going. <module-name> is the name you wish to be assigned to the executable program module created by TURNKEY. After you press the carriage return, TURNKEY will start the process of turning your program into a stand-alone application. When TURNKEY is finished it will return to the shell and your application should be visible if you perform a directory listing of your current execution directory.

TURNKEY Examples

For an example of a simple TURNKEY application, please refer to the TURNKEY glossary page in the back of this manual.

How TURNKEY Works

TURNKEY first creates a new executable file and copies only the MACH2 kernel code (approximately 5K bytes) to the file. Next, TURNKEY takes all code found in the user code segment and writes it also to the executable file. Finally, TURNKEY performs all the actions required to have OS-9 recognize the file as an executable program module (fixing the CRC, preparing the module header, etc.). Since the MACH2 compiler naturally generates relocatable code (PC-relative), the executable program modules generated by TURNKEY will be classified as 'position-independent'.

VERBOSE, a TURNKEY Utility

To ensure that your application program does not contain any compiled references to MACH2 compiler words (since the compiler words are not included in a TURNKEY application) you should place a positive value in the system variable 'VERBOSE' before loading your program. The next time you load your program, any illegal references to compiler words will be flagged with error messages.

ABORT Considerations

The default run-time code executed during abort handling (when your program uses the word ABORT or ABORT") is the word QUIT. QUIT is a compiler word and thus cannot be included in TURNKEY applications. If you do wish to use ABORT or ABORT" in your application be sure to install a custom abort handling routine (see the ABORT glossary entry).

Notes:

34

TURNKEY

Development Tools

THE MACH 2 INTERACTIVE ASSEMBLER

An Interactive Assembly Environment for 68000 Programmers

MACH 2 can be used exclusively as an interactive assembly language development environment for those assembly-language programmers who'd rather not write in FORTH. The assembler uses completely standard syntax and attempts to follow the syntax of the OS-9/68000 Assembler wherever possible. New subroutines can be typed in from the keyboard or loaded in from files. By merely typing the name of a subroutine it will be executed. Each and every subroutine in your program can be tested and debugged symbolically and interactively, one at a time. The entire program need not be run in one fly-or-die pass.

The most notable aspect of the MACH 2 Assembler is that it is NOT the RPN-format assembler usually included in most FORTH systems. It is a standard, infix 68000 assembler. 68000 assembly examples may be copied straight from most references without alterations. All the mnemonics are standard (see the appendix for a list of them).

The increase in readability and supportability of assembly language routines written in the MACH 2 assembler over assembly routines written in RPN assemblers is dramatic.

FORTH and Assembly Language

Why is an assembler needed with FORTH?

> Speed and control. MACH 2 runs a bit slower than hand-optimized code would. Also, there is no explicit access to the actual CPU from MACH 2. With a just a little knowledge of how FORTH runs, programmers can take control of the 68000 and still work within an interactive, development environment (MACH 2).

What is the advantage of FORTH then?

> FORTH can be thought of as a library of pre-written assembly language routines. This library is at the programmer's fingertips and allows lightning-fast application development. In fact, these FORTH routines already do many of the common things assembly language programmers will want to do in their programs (move blocks of memory, switch parameters on the stack, display characters on the screen, take numeric input according to a base).

> Customized assembly language routines should only be used to tighten up FORTH code or to perform hardware-related tasks where speed is of the utmost importance.

The following pages provide a brief introduction to the MACH 2 interactive assembler. The appendix contains additional information. For more information on 68000 assembly language programming, see the bibliography for recommended reference readings.

MACH 2 Interactive Assembler

USING THE MACH 2 INTERACTIVE ASSEMBLER

CODE Definitions

The major difference between writing assembly code in MACH 2 and writing assembly code in a stand-alone assembler is that in MACH 2 you must surround all of your assembly subroutines and code fragments with the words CODE and END-CODE. CODE is used to name the routine and END-CODE is used to mark the end of the routine:

```
      CODE 4/  ( n -- n/4 )   ( CODE makes a dictionary header for a new word, )
          MOVE.L  (A5)+,D0     ( '4/', and starts compiling. The words between )
          ASR.L   #2,D0        ( CODE and END-CODE are run when '4/' is executed. )
          MOVE.L  D0,-(A5)     ( From the stack notation, a number should be on the )
          RTS                  ( stack. That number will be divided by four and )
                               ( returned on the stack. END-CODE signals the end )
      END-CODE                 ( of the definition and stops compilation. )
```

The word CODE puts FORTH into the compilation mode, adds the ASSEMBLER vocabulary to the search order, and installs an ABORT vector to handle local branching.

The word END-CODE is like ';' except that ';' will automatically compile an RTS ($4E75) into the definition being constructed. In a code definition this must be done explicitly by using the RTS mnemonic. This is an important point. In addition, the current number base is saved by CODE and restored by END-CODE. During compilation the base is decimal.

Assembler mnemonics such as MOVE.L are immediate compiling words whose job it is to parse the input stream and compile the correct op-codes into the dictionary. Only later, when the word the assembly language instruction was compiled into is run, will the 68000 actually run a MOVE.L instruction.

Referencing Previous Definitions.

Two methods are available for making references to other definitions. Either use the name by itself (since the system is in compile mode while in code definitions, the FORTH compiler will do the compiling):

```
      CODE Example1  ( n1 n2 -- n3  )
          OVER
          SWAP
          *
          RTS
      END-CODE
```

...or by using the assembly language 'jump to subroutine' instruction: 'JSR <name>' (in this case the assembler will compile <name> without regard to immediate or macro bits--see the macro discussion which follows):

```
CODE Example2 (  n1 n2 -- n1 n2 n1   )
    JSR  OVER
    JSR  SWAP
    JSR  "*"
    RTS
END-CODE
```

If the above method is used, any FORTH words which contain characters which are also used as arithmetic operators in assembler expressions (* , / , - , /MOD, FORTH-83, etc.), must be surrounded by quotation marks (see "*" in the above example).

Making OS-9 System Calls from Assembly

To make an OS-9 system call from within a CODE definition, use the 'OS9' assembler psuedo-instruction:

```
CODE SendSignal ( processID signal -- errorcode )
    MOVE.L  (A5)+,D1       \ get signal to send from stack
    MOVE.L  (A5)+,D0       \ get intended receiver's process ID from stack
    OS9     F$Send         \ make the F$Send call to the OS-9 kernel module
    BCC.S   @NOERR         \ if carry bit clear then no error occurred

    EXT.L   D1             \ otherwise, extend the word-length error code
    MOVE.L  D1,-(A5)       \ place the error code on the parameter stack
    BRA.S   @EXIT          \ always place an RTS at the end of a subroutine

@NOERR
    CLR.L   -(A5)          \ return zero if no error exists

@EXIT
    RTS                    \ return from subroutine
END-CODE
```

There are many important things to notice in this example. First, the line 'OS9 F$Send' is equivalent to the following assembly language sequence:

```
TRAP  #0
DC.W  6
```

Trap #0 is the software exception vector used to access the system calls in the OS-9 kernel module. The word-length data following the trap call is the selector code used to identify which routine in the kernel is being called. A listing of the selector codes corresponding to all available OS-9 system calls is included in an appendix.

The register usage for all of the OS-9 system calls is listed in chapters 14, 15, and 16 of the OS-9/68000 Operating System Technical Manual. Many of the system calls will return, as this call did, with the carry bit set if an error occurred. Finally, note that it is very easy to interact with the MACH2 parameter stack.

MACH 2 Interactive Assembler

ASSEMBLER SYNTAX

MACH 2 code definitions may contain some or all of the following:

- instructions (68000 assembly language instructions or FORTH words) separated by at least one space or psuedo-instructions (.ALIGN, DC, etc.)

- local labels

- comments

Local Labels

Local labels are only valid within the CODE definition in which they are used. Local labels must be in the form @xxx where xxx is any ascii string which does not contain spaces. Only 16 forward references may be made to any single local label but any number of back-references are allowed.

The definition of the FORTH word 0= uses a local label for a forward branch:

```
CODE 0=   ( n  -  f )

    MOVEQ.L  #0,D0
    TST.L    (A5)+
    BNE.S    @1
    MOVEQ.L  #-1,D0
@1  MOVE.L   D0,-(A5)
    RTS
END-CODE
MACH
```

Instructions and Pseudo-Instructions

An instruction can be a 68000 instruction or a previously defined FORTH word. 68000 instructions are described in the 68000 Reference Manual, Fourth Edition. Explanations of MACH 2 FORTH words are in the glossary. If an instruction requires an operand, at least one space should separate the instruction and operand.

A pseudo-instruction is an assembler instruction which generates code but is not actually a 68000 instruction. Pseudo-instructions help improve program readability. The pseudo-instructions provided by the MACH2 assembler are described later in this chapter.

Comments

The standard FORTH commenting words '(' and '\'should be used for comments in assembly language routines. See the '(' and '\' glossary entries. The '(' commenting word supports nested comments.

ASSEMBLER EXPRESSIONS

Addressing modes and assembler directives often use expressions as part of their operands. Numbers, and symbols that represent numbers can be used in expressions.

Numbers

Three types of numbers may be used: decimal, hexadecimal, and binary.

308	Decimal numbers are the default.
$3FC	Hexadecimal numbers must be preceded by a '$'.
%110	Binary numbers must be preceded by a '%'.

Operations

The MACH 2 assembler supports the following arithmetic, shift, and logical operations in an expression:

Type	Operation	Operator	Comment
Arithmetic	Addition	+	
	Subtraction	-	
	Multiplication	*	
	Division	/	Integer result
	Negation	-	
Shift	Shift Right	>>	Zeros shifted in
	Shift Left	<<	Zeros shifted in
Logical	And	&	
	Or	!	

Operator Precedence

Multiple operators in an expression are evaluated in the following order (operators with the same precedence are evaluated from left to right) :

1. Operations in parentheses (innermost parentheses evaluated first).
2. Negation.
3. Shift operations.
4. Logical operations.
5. Multiplication and division.
6. Addition and subtraction.

As the following example demonstrates, the use of inline math in assembly code can improve program readability:

```
MOVE.W #1<<8+42,D1          MOVE.W #298,D1
OS9     F$PErr      VS      OS9     F$PErr
```

Symbols

In the MACH 2 assembler, a symbol is a string used to represent either a number or a complete effective addressing mode. Numbers are assigned to symbols with the defining word CONSTANT or the assembler directive EQU. A symbol is assigned to an effective addressing mode with the EQU directive. Any character may be used in a symbol string except for the following: '#' , ',' , '(' , ')' , the digits 0 - 9, and the arithmetic operators listed on the previous page. Do not use the names of any predefined FORTH or ASSEMBLER words (e.g. TYPE, A0) for symbols.

ASSEMBLER DIRECTIVES

EQU , A Symbol Definition Directive

The EQU directive is used to assign a numerical value, expression, or an addressing mode to a symbol.

```
EQU   12   Motor3Offset              \ using EQU to assign the number
                                      \ 12 to the symbol Motor3Offset,
EQU   3*4   Motor3Offset             \ using this expression with EQU would
                                      \ also work
EQU   Motor3State   Motor3Offset(A0)  \ using EQU to assign the addressing
                                      \ mode 12(A0) to the symbol
                                      \ Motor3State
```

The use of symbols with assembly language tends to produce much more readable and meaningful source files. For example, both of the following instructions would obtain the the value which indicates the current state of motor #3 from an array of several motor state flags. The instruction which uses a symbol gives a better idea of what is going on:

```
With symbols:        MOVE.L        Motor3State,D0
Without symbols:     MOVE.L        12(A0),D0
```

When the string 'Motor3State' is compiled it will be replaced with the addressing mode 'ioComp(A0)' where 'ioComp' is the constant 12.

DS, A Data Storage Allocation Directive

The DS directive is used to reserve space for variables in the variable space.
Length is an expression which specifies the number of bytes, words, or long words to be reserved :

```
LABEL   <name-of-space>   DS.B   length
LABEL   <name-of-space>   DS.W   length
LABEL   <name-of-space>   DS.L   length

Ex:     LABEL   CurrentTime   DS.L   1
```

41

MACH 2 Interactive Assembler

The above ex~ ~ole is equivalent to the following use of the FORTH word VARIABLE:

        VARIABLE <name-of-space>   (CellsDesired) (size) * 4 - VALLOT

Executing the name of the storage space will put its address on the stack, just as in FORTH.
Examples of accessing DS storage areas from assembly:

        MOVE.L   MYVARIABLE,D0   (fetch)
        MOVE.L   D0,MYVARIABLE   (store)

ASSEMBLER PSEUDO-INSTRUCTIONS

Data Allocation Pseudo-Instructions - DC , DCB

In a MACH 2 program, the memory space is divided up into two parts: the code space
and the variable space. Define Constant (DC) and Define Constant Block (DCB) are
used to define constant data which is located in the code space (program area).
Define Storage (DS) is used to allocate space for variables in the variable space (the
uninitialized data area). The MACH2 assembler has no provisions for allocating data
in the initialized data area.

DC (Define Constant)

The DC pseudo-instructions below will place data in the code space. The three different
forms of the DC pseudo-instructions generate data which is either byte aligned (DC.B), word
aligned (DC.W), or long word aligned (DC.L). The DC.W and DC.L pseudo-instructions will
always align their data on word or long word boundaries, respectively.

        HEADER   <name-of-constant>   DC.B   value(s)
        HEADER   <name-of-constant>   DC.W   value(s)
        HEADER   <name-of-constant>   DC.L   value(s)

        Ex: HEADER   MotorOn   DC.B $A

Multiple values should be separated by commas. A '$' must be used before a hexadecimal
value. (Note that DC.B $A is equivalent to HEX A C,) A '%' must be used before a binary
value. Arithmetic expressions may also be used with DC.

The DC.B directive may be used to lay strings into memory as follows. Note that the string should be delimited by single quotes. To define a string constant which contains a single quote the single quote must be preceded by a single quote. .ALIGN should be used after the definition of a string constant to ensure that the dictionary pointer ends up on an even word boundary.

```
HEADER    CountedString    DC.B    5,'Hello'
                           .ALIGN
HEADER    C-String         DC.B    'Hello',0
                           .ALIGN
HEADER    StringWithQuote  DC.B    'Don"t'
                           .ALIGN
```

DCB (Define Constant Block)

The DCB pseudo-instruction is used to reserve blocks of memory in the code space that are to be initialized to a certain value. Length specifies the number of bytes (DCB.B), words (DCB.W) or long words (DCB.L) in the block. Value specifies the value to be stored in the bytes, words, or long words which comprise the block.

```
HEADER    <name-of-data-block>    DCB.B    length,value
HEADER    <name-of-data-block>    DCB.W    length,value
HEADER    <name-of-data-block>    DCB.L    length,value

Ex:    HEADER   ThreeSpaces   DCB.W   3,$20
```

The above example is equivalent to a HEX 20 W, 20 W, 20 W, .
The address of a data block is also obtained by 'ticking' its header.

Examples of accessing DC(B) data from assembly language:

```
CMP.B    MotorOn,D0      ( Compares what is in the D0 to $A )
MOVE.B   MotorOn,D0      ( Moves the byte value $A into the D0 )

LEA      ThreeSpaces,A0  ( Puts the address of ThreeSpaces in the A0 register )
```

The TCALL and OS9 Pseudo-instructions

The OS9 pseudo-instruction was discussed previously. The TCALL pseudo-instruction is used to generate calls to user trap handler modules. TCALL's syntax and an example of its use are shown below:

```
TCALL    <trap vector#>,<function code>

CODE CallTrapModule ( - )
     TCALL   3,8
     RTS
END-CODE
```

43

MACH 2 Interactive Assembler

The <trap vector#> is the number of the 68000 software exception vector (#0-15) used to access the trap handler module (refer to the section on user trap handler modules for more information). <function code> is a value which will be passed from the calling program to the trap module. The TCALL pseudo-instruction will generate the following assembly language sequence:

```
TRAP   <trap vector#>
DC.W   <function code>
```

Note that the definition of the OS9 pseudo-instruction is TCALL 0,<selector> .

MACH 2 MACROS

MACH 2 allows macro substitution (see the discussion in the appendix). One bit in the header of a word is reserved as the 'MACH' bit. If a word with its MACH bit set is encountered during compilation, all of the assembly language instructions which comprise the word will be laid into the definition being built. Normally, a JSR to the executable code for the word would be compiled. This technique can be used to decrease execution time of words and, at times, save space.   Both colon and code definitions may be marked as macros.

To create a FORTH macro use the word 'MACH' after the code definition. The word DUP has been defined as a MACH 2 macro word for two reasons. One reason is that the opcode which does a DUP takes up half the memory as the opcode for a jump-to-subroutine instruction. The second reason is that it is much faster to execute in-line code than to jump to a routine in a separate location and then return.

```
CODE  DUP
      MOVE.L   (A5),-(A5)
      RTS
END-CODE    MACH
```

Three precautions should be observed when setting the MACH bit on a word :

1)   Only the instructions up to the first RTS will be transferred so make sure the routine to be 'macro-ed' has only one RTS and that the RTS is in the last line in the routine.

2)   Any routines which contain PC-relative references to words outside of the current code definition will not run correctly if moved into a definition in a different location.

3)   Excessive use of MACH words may yield a large increase in program size with only a small speed improvement.

MACH sets bit 6 in the count byte of the name field (see the dictionary header appendix). It works exactly like the words IMMEDIATE and SMUDGE.

44

MACH 2 Interactive Assembler

THE MACH 2 SYMBOLIC DISASSEMBLER

A disassembler could be described as the static equivalent of a debugger. The debugger lets you observe your code dynamically, as it runs. The disassembler lets you look at your code statically, it lets you see the way your code has been put into memory. If your program is written in high-level FORTH, the disassembler lets you see the assembly language instructions the MACH 2 compiler used to implement your FORTH words.

A disassembler performs the opposite function of an assembler. When a program is assembled, all of the human-readable commands (such as MOVE.L D0,D1 , or JMP (A0)) are converted to the binary numbers which the computer understands. The human-readable commands are called instruction mnemonics and their corresponding numerical values are called opcodes. A disassembler will take the opcodes generated by an assembler and convert them back to the instruction mnemonics which a person understands.

Symbolic Disassembly

The MACH 2 disassembler takes the disassembly process one step further by also listing the name of the FORTH word which is currently being disassembled to the right of the corresponding assembly instruction. This helps you keep track of where you are in a particular word. The action of associating a name (symbol) with an instruction while disassembling is called symbolic disassembly.

IL: The MACH2 Disassembly Command

To access the MACH 2 symbolic disassembler use the 'IL' disassembly command . IL is located in the FORTH vocabulary. IL will disassemble 'count' instructions starting at the specified address. The names of the routines being disassembled will also be listed in-line with the instructions being displayed.

        start-address   count   IL

        Ex: ALSO ASSEMBLER <cr> ok <0>
                HEX <cr> ok <$0>
                10000 20 IL <cr>

The above example will disassemble the 32 instructions starting at address 10000 hex. If there are no valid instructions at the address provided, the disassembler will still attempt to convert the numbers it finds into instruction mnemonics. The disassembly will appear random.

Note: The MACH2 disassembler code is located in a user trap module which is accessed by MACH2 through software exception vector #13.

THE MACH 2 SYMBOLIC DEBUGGER

When your program is not running correctly and you can't understand why or, when you want to step through your code JUST to make sure its doing what you think it is, it's time for the debugger.

A debugger is a tool which lets you watch your program run, assembly instruction by assembly instruction. After each instruction executes you can examine or change the contents of the registers, memory, or FORTH stacks. If you would only like to see a small section of your code execute you can insert a break point (a STOP! command) at the beginning of the section of interest and then run your program normally. When the breakpoint is encountered the debugger will stop execution of your program and display the contents of all 16 68000 registers and the assembly language instruction about to be executed.

Invoking the Debugger

In order to use the debugger, you must somehow 'invoke' or 'call' it. There are two ways of entering the debugger:

 # 1   Compile the DEBUG command into a definition

If you include the word DEBUG in one of your definitions, the debugger will be entered whenever you execute that definition.

    : TEST DEBUG 100 DO I . LOOP ;

If you were to execute TEST, the debugger would stop execution right before the DO...LOOP.

 # 2   Execute the DEBUG command Interactively

If you interactively execute the word DEBUG, the debugger will stop the execution of MACH 2 immediately:

    DEBUG <cr>

If you enter the debugger this way you will not know where the debugger has stopped. You will have to use other debugger commands to get to the section of your program you wish to examine.

If you interactively execute DEBUG followed by the name of a pre-defined word, the debugger will immediately set a break at the first instruction in the word. The next time the word is executed, the debugger will be entered:

    : GO 2 3 DUP ;
    DEBUG GO <cr>

The Debugger Display

The following picture shows the display which is generated when you enter the debugger:

```
Assembly language instruction
about to be executed.  This is the
first Instruction in the word GO.                      FORTH word about
                                                       to be executed.
                                    Flag settings.

DEBUG GO
00055232:    MOVE.L        (A5),-(A5)                      DUP
PC: 00055232    SR: 0310      X=1 N=0 Z=0 V=0 C=0
A0: 00051154    A4: 00000000  D0: 00055232   D4: 00000008
A1: 00051152    A5: 0001F1E6  D1: 00000007   D5: 00000007
A2: 00000000    A6: 00021100  D2: 00000003   D6: 00000007
A3: 0001EA32    A7: 0002017E  D3: 00000000   D7: 0001EA0A

> A0 567E2
                    Changing the
                    contents of the      This display shows the
                    A0 register          contents of all registers
                                         prior to the execution
This is the debugger                     of DUP.
prompt.  It means we
are in the debugger
and that the debugger
is awaiting a command.
```

Once you have the debugger prompt you can use any of the debugger commands listed below to move around in the debugger.

Important Debugger Information !!

As the debugger works, it will write data into the code area of the program being debugged and in the MACH2 program module. This means that while the MACH2 debugger is in use, it will be generating self-modifying code. Since self-modifying code can have disastrous consequences in a multi-tasking, multi-user system such as OS-9, you must make sure that when you use the MACH2 debugger, you are the only person using MACH2 on your system.

THE DEBUGGER COMMANDS

The following pages list all of the debugger commands at your disposal.

An   <expr>

Modify or display the contents of address register n. If 'An' is followed by an expression, the specified register's contents will be changed to the value of the expression. If 'An' is not followed by an expression, the current contents of the specified address register will be displayed:

```
    >  A1   <cr> 51152        \ Display the contents of register A1
    >  A1   RA6+8<cr>         \ Set the contents of register A1 to the contents of
                              \ register A6 plus 8.
```

BR   <addr>   <cnt>

Used to set or display normal or counted breakpoints. If used without any of the optional parameters, 'BR' will display the addresses of all temporary breakpoints currently set. For counted breakpoints, 'BR' will also display a count of how many more times the breakpoint must be hit before program execution is halted. If 'BR' is followed by an address, a breakpoint will be set at the address. If 'BR' is followed by an address and a count, a counted breakpoint will be set at the address. A counted breakpoint is a breakpoint which must be hit <cnt> times before program execution will actually be suspended. Each time a counted breakpoint is hit, the registers will be displayed but program execution will not stop until the count reaches 0.

```
    >  BR   54112   <cr>
    >  BR   54120   8  <cr>
    >  BR   <cr>
    54112
    54120 8
```

CL   <addr>

Clear break points. 'CL' followed by an address will remove the breakpoint at that address. 'CL' followed by no address will remove all temporary break points.

CV   EXPR

Convert expression. 'CV' must be followed by an expression. The expression may contain any of the arithmetic operators described in the assembler section, binary(%)/octal(^)/decimal(#)/hexadecimal($) numbers, and register operators. 'CV' will convert the expression to its hexadecimal and decimal equivalent values.

```
    >  CV   RA6+%1010+$4E+#400+^55   <cr>
       Hex:21315   Dec:135957
```

DM <addr> <cnt>

Display the contents of memory. If no parameters follow 'DM', 16 bytes of memory, starting from the address last used with any debugger command, will be displayed. If an address follows 'DM', the 16 bytes of memory starting at the address will be displayed. If an address and a count follow 'DM', the 'count' bytes of memory starting at the address will be displayed.

A convenient way to specify a register address to 'DM' (or any other debugger command which accepts an address) is to place an 'R' in front of the desired register symbol (i.e. use 'RA7' to specify the address in register A7). To add a level of indirection, precede the 'R' with one or more '@' signs:

```
> DM   RA7  <cr>          \ Display memory starting at the address in register A7.
02017E:   0006 06A4 0006 079A 0005 D544 CODE FEED   ...........D....

> DM   @RA7 10  <cr>   \ Display memory at the address located at the address in A7.
0606A4:   4EBA D786 4EBA D610 4A92 671C 6110 0E20   N...N...J.g.a...
0606B4:   5374 6163 6B20 456D 7074 7920 2120 2B1F   Stack.Empty.!.+.
```

Dn <expr>

Modify or display the contents of data register n. See 'An'.

E S

Exit to the OS-9 shell.

G

Go. Continue program execution until the next break point is encountered.

GT ADDR

'Goto'. Sets a temporary breakpoint at the address. Then, program execution is continued until the breakpoint is reached. When the breakpoint is reached, it is automatically removed. Useful for 'walking through' a program.

H

Help. Displays a summary of the available debugger commands. Optional command parameters are places between arrows (i.e. <addr>). Required command parameters are capitalized (i.e. ADDR).

MACH 2 Symbolic Debugger

```
> H <cr>
```

An	<expr>	Modify/display address register
BR	<addr> <cnt>	Set/display breakpoints
CL	<addr>	Clear breakpoint
CV	EXPR	Convert expression
DM	<addr> <cnt>	Display memory
Dn	<expr>	Modify/display data register
ES		Exit to shell
G		Go/run
GT	ADDR	Goto temporary breakpoint
IL	<addr> <cnt>	Instruction list
PC	<expr>	Modify/display program counter
RX		Toggle register listing
S	<cnt>	Single step
SB	ADDR EXPR	Set memory [byte]
SW	ADDR EXPR	Set memory [word]
SL	ADDR EXPR	Set memory [long]
SR	<expr>	Modify/display status register
SS		Show parameter stack
T	<cnt	Trace instr. TRAP/JSR = 1
TD		Display all registers
C N V X Z		Modify/display status bits

IL <addr> <cnt>

Instruction list. Disassembles <cnt> instructions starting at <addr>. If no count is specified, the 10 instructions starting at the address are disassembled. If no address or count are specified, the 10 instructions starting at the last address used by a debugger command are disassembled. *Note that 'IL' is both a debugger command and a FORTH word and that it is used differently in both environments.*

PC <expr>

Modify or display the contents of the program counter. See 'An'.

RX

Toggle register display. Toggles between a complete register display and a display of only the current instruction.

S <cnt>

Step. Execute one assembly language instruction and stop. If a count is specified, execute 'count' assembly language instructions and stop.

Interfacing to OS-9

Notes:

52

MACH 2 Symbolic Debugger

EXPECT Accepts Line Editing Commands

The FORTH word EXPECT is the main word used by MACH2 as it interactively accepts lines of user input. As the following assembly language definition shows, EXPECT uses the I$ReadLn I/O system call and, therefore, supports the standard OS-9 line editing commands:

```
CODE  Expect  (  a  n  -  )
    MOVE.L   (A5)+,D1         \ length
    MOVEA.L  (A5)+,A0         \ buffer address
    MOVEQ.L  #0,D0            \ I/O channel
    OS9      I$ReadLn         \ read in the string
    BCS      OS9_ERROR        \ MACH2 error handling word
    SUBQ.L   #1,D1            \ set up SPAN, see the SPAN
    MOVE.L   D1,SPAN(A5)      \ glossary page.
    RTS
END-CODE
```

KEY Does Not Accept Line Editing Commands

The FORTH word KEY is the word used by MACH2 when it takes single character input from the user. Since KEY uses I$Read to read in single characters it does not support the use of the OS-9 line editing commands. KEY's function is to return the ASCII code corresponding to the input character:

```
CODE  KEY  (  -  c  )
    JSR      Set_NoEcho       \ turn echo parameter off for this terminal
    CLR.L    -(A5)
    LEA      3(A5),A0         \ I$Read will return the character on the stack
    MOVEQ.L  #0,D0            \ use default path
    MOVEQ.L  #1,D1            \ read only one byte
    OS9      I$Read           \ Read !
    BCS      OS9_ERROR        \ MACH2 error handling word
    JSR      Reset_Echo       \ turn echo back on for this terminal
    RTS
END-CODE
```

The 'Set_NoEcho' and 'Reset_Echo' subroutines are shown on the following pages as examples of changing a terminal's operating characteristics. Note that although KEY will not recognize the standard line editing command keys it will still be affected by the special "interrupt" keys (CONTROL C and CONTROL E). The section on OS-9 signals explains how "interrupt" keys are processed.

55

The OS-9 Terminal Device

Getting/Setting Terminal Characteristics

The list below shows the terminal characteristics which may be changed interactively, by using either the TMODE (see page 6-107 of the OS-9/68000 OS User's Manual) or XMODE (see page 6-119 of the OS-9/68000 OS User's Manual) utility commands, or from within a program by using the I$GetStt and I$SetStt (see pages 15-9 and 15-18 of the OS-9/68000 OS Technical Manual) I/O system calls:

Letter case
Destructive backspace
Line delete
Echo
Automatic line feed
End of line null count
End of page pause
Page length
Backspace "input" character
Delete line character
End of record character
End of file character
Reprint line character

Duplicate last line character
Pause character
Keyboard interrupt character
Keyboard abort character
Backspace "output" character (echo char)
Line overflow character
Parity code, # of stop bits & bits/char
Software adjustable baud rate
X-on character
X-off character
Tab character
Tab field size

TMODE is used to set/check the operating characteristics of the terminal currently in use. Changes made with TMODE will only remain in effect until the path to the user's terminal is closed. XMODE is used to set/check the default characteristics which will be given to all terminal paths which are subsequently opened (XMODE actually updates the memory image of the device descripter information for terminals). Changes made with XMODE will remain in effect until the computer is shut down. Refer to the XMODE utility command description on page 6-119 of the OS-9/68000 Operating System User's Manual for information on how to permanently change terminal characteristics.

Interactively Checking a Terminal's Operating Characteristics

The following example demonstrates how TMODE may be used interactively from within MACH2 to check the operating characteristics of the terminal currently being used by MACH2 (if no arguments follow TMODE it will display the current terminal operating characteristics):

```
$   TMODE   <cr>

/term
noupc  bsb  nobsl  echo  lf  null=0  nopause  pag=24  bsp=08  del=18  eor=0D
eof=1B  reprint=04  dup=01  psc=17  abort=03  quit=05  bsc=08  bell=07
type=00  baud=9600  xon=11  xoff=13  tabc=09  tabs=4
 ok <0>
```

The OS-9 Terminal Device

The table on pages 6-107 through 6-110 of the OS-9/68000 OS User's Manual will help decipher the MACH2 terminal characteristics returned by TMODE.

Setting a Terminal's Operating Characteristics from within a Program

The two subroutines shown below show how a terminal's operating characteristics may be altered from within a running program. These are the two subroutines used by KEY in a previous example. Since the FORTH standard requires that KEY does not echo its input, the subroutine 'Set_NoEcho' is used to turn off echoing in the current terminal path. After the character has been input, 'Reset_Echo' is used to turn character echo back on.

The I$SetStt I/O system call used to alter terminal characteristics allows you to change multiple terminal characteristics at once. Typically, the process used to change a terminal characteristic involves: 1. using I$GetStt to read in a table containing the current terminal characteristics; 2. altering the field or fields in the table which correspond to the characteristics you wish to change, and 3. using I$SetStt to write the contents of the altered table of information out to the path process descriptor for the terminal.

```
        EQU  SS_Opt 0                 \ selector used with I$GetStt and I$SetStt
                                      \ I/O system calls

        CODE   Set_NoEcho   (  -  )
            LEA        -128(A7),A7    \ allocate a 128 byte buffer on the system stack
            MOVEA.L A7,A0             \ put the address of the start of the buffer in A0
            MOVEQ.L #0,D0             \ use default path
            MOVEQ.L #SS_Opt,D1        \ 'read in path descriptor options' selector
            OS9        I$GetStt       \ get terminal characteristics
            BCS        BYE_ERROR      \ internal MACH2 error handling routine

            MOVEA.L A7,A0             \ put start addr of terminal char. table in A0
            MOVE.W  4(A0),MODE(A6)\   record the echo and linefeed settings
            CLR.W    4(A0)           \ set no echo and no linefeeds
            MOVEQ.L #0,D0             \ use default path
            MOVEQ.L #SS_Opt,D1        \ 'write path descriptor options' selector
            OS9        I$SetStt       \ set terminal characteristics
            BCS        BYE_ERROR      \ internal MACH2 error handling routine
            LEA        128(A7),A7     \ de-allocate the stack space
            RTS
        END-CODE
```

```
CODE    Reset_NoEcho    ( - )
    LEA       -128(A7),A7      \ allocate a 128 byte buffer on the system stack
    MOVEA.L   A7,A0            \ put the address of the start of the buffer in A0
    MOVEQ.L   #0,D0            \ use default path
    MOVEQ.L   #SS_Opt,D1       \ 'read in path descriptor options' selector
    OS9       I$GetStt         \ get terminal characteristics
    BCS       @1               \ exit if error

    MOVEA.L   A7,A0            \ put start addr of terminal char. table in A0
    MOVE.W    MODE(A6),4(A0)   \ restore saved echo and linefeed settings
    MOVEQ.L   #0,D0            \ use default path
    MOVEQ.L   #SS_Opt,D1       \ 'write path descriptor options' selector
    OS9       I$SetStt         \ set terminal characteristics
@1  LEA       128(A7),A7       \ de-allocate the stack space
    MOVE.W    #1,MODE(A6)      \ -1 in mode means 'echo' is not currently altered
    RTS
END-CODE
```

The OS-9 Terminal Device

FILES

FORTH File Handling Words

MACH2 contains two sets of file handling words. One set of words is comprised of the file handling words built into the FORTH language (BLOCK , BUFFER , VIRTUAL , UPDATE , SAVE-BUFFERS , EMPTY-BUFFERS , LIST , LOAD , FLUSH). Since MACH2 has been designed to edit and load text files, as opposed to block files, these block-oriented FORTH file handling words have been included in MACH2 only to maintain FORTH-83 compatibility. All of these FORTH file handling words are found in the FORTH vocabulary. For more information on these words, refer to their individual glossary pages.

MACH2/OS-9 File Handling Words

The other set of file handling words contained in MACH2 are designed for easy interaction with the OS-9 Random Block File Manager (RBF). These are the words which will be discussed in detail below. All of the OS-9/MACH2 file handling words are located in the OS-9 vocabulary.

Creating Files

The word $CREATE is used to create and open new files. The stack notation for $CREATE is shown below:

$CREATE (path-name attributes access-mode - path# errorcode)

'path-name' should be the address of a null-terminated string which contains the pathname to be uses to find the new file. 'attributes' is a value which determines how and by whom the new file may be accessed in the future; it defines all the possible ways the file may be interacted with in the future. 'access-mode' is a value which indicates the current access permission desired; it indicates how we would currently like to interact with the file.

The table below shows how to choose the values used for the attributes parameter:

Attribute Bits	Set this bit...	with this value...	to permit this type of access.
	0	1	owner read permit
	1	2	owner write permit
	0+1	3	owner read/write permit (owner update)
	2	4	execute permit
	3	8	public read permit
	4	16	public write permit
	3+4	24	public read/write permit (public update)
	5	32	public execute permit
	6	64	non-sharable file

59

Files

Mode Bits	Set this bit..	with this value..	If you desire this type of access.
	0	1	read
	1	2	write
	0+1	3	read/write (update)
	2	4	execute
	3	8	...
	4	16	...
	3+4	24	...
	5	32	...
	6	64	single-user

The example below shows how to use $CREATE to create a new file named 'MyFile' (which will be located in the current directory since no additional pathname specifications were included with the filename), which may be read from or written to by anyone and will be opened in a read/write mode for current access:

```
ONLY    FORTH    DEFINITIONS
ALSO    OS-9                        \ OS-9 file words are located in the OS-9 voc.

BINARY                              \ put in BINARY base for readability
0000011  CONSTANT   OwnerR/W        \ set bits 0 and 1 for owner read/write access
0011000  CONSTANT   PublicR/W       \ set bits 3 and 4 for public read/write access
0000011  CONSTANT   R/WMode         \ set bits 0 and 1 for read/write access mode
DECIMAL                             \ return to DECIMAL base

"  MyFile"  1+                      \ add 1 to file name address to skip length byte
OwnerR/W   PublicR/W   +            \ this file may have owner r/w and public r/w
R/WMode                             \ right now we want r/w access
$CREATE  <cr> ok <2>                \ create the file, $CREATE returns 2 values

.S   <cr> 3 0 <- TOP ok <2>         \ top item is error code, second is path number
```

As the example shows, $CREATE will return two values on the stack. The top item will be an error code. If the error code is 0, no error occurred. If the error code is non-zero it is an OS-9 error number (you may want to pass the error code to the ?OS9ERROR error handling word, described in the next section). The second item, if no error occurred, will be a path number since $CREATE will leave the new file open. The path number is usually required by all OS-9 file routines which operate on open files. A path number is used to uniquely identify and locate an open file.

Opening Files

$OPEN is used to open files. It has the following stack notation:

```
$OPEN  (  path-name  access-mode  -  path#  errorcode  )
```

60

Files

'path-name' should be a null-terminated string which contains the path specifications which should be used to find the file to be opened. The 'access-mode' parameter is the same parameter described in the $CREATE discussion. Here is an example use of $OPEN:

    " Monday/Work_File" 1+   R/WMode $OPEN <cr> ok <2>

    .S <cr> 4 0 <- TOP ok <2>

The above example shows how a file named 'Work_File', which is located in the 'Monday' directory, could be opened with read/write access mode permission. $OPEN returns two values on the stack. The top value is an error code an the second value, if no error occurred, will be the path number which uniquely identifies the open file.

Closing Files

To close a file use $CLOSE. $CLOSE expects to be passed the path number of the open file it is to close. $CLOSE will return an error code on the parameter stack. Here is the stack notation for $CLOSE, and an example showing how the file opened above could be closed:

    $CLOSE (path# - errorcode)

    4 $CLOSE . <cr> 0 ok <0>

Deleting Files

$DELETE is used to delete files. $DELETE expects to be pass the address of a null-terminated string which contains the pathname used to locate the file to be deleted. The user deleting the file must have non-sharable write access and the file must be closed. The access mode is used to specify the data or execution directory (but not both) in the absence of of a full pathlist. If the access mode specified is read, write, or update, the current data directory is assumed. If the execute bit is set in the access mode parameter, the current execution directory is assumed. Note that if a full pathlist is given (a full pathlist begins with a '/'), the access mode parameter is ignored. The stack notation for $DELETE and an example of its use are shown below:

    $DELETE (path-name access-mode - errorcode)

    " N._File" 1+ R/WMode $DELETE . <cr> 0 ok <0>

61

Files

Writing Data to a File

To write data to an open file use $WRITE. $WRITE should be passed the address of the buffer where the data to be written resides, the number of bytes which should be written, and the path# which identifies the file to which the data should be written. $WRITE will return two values on the stack when it has finished. The number on top of the stack will be an error code. The second value will be the number of bytes actually written.

The file to be written to must have been opened with read/write (update) or write access mode permission. If the data is written past the current end-of-file, the file will automatically be expanded. The stack notation for $WRITE and an example of its use are shown below:

```
$WRITE  ( bufferaddr len path#  -  #byteswritten errorcode )


ONLY   FORTH   DEFINITIONS  <cr> ok <0>
ALSO   OS-9  <cr> ok <0>

BINARY  <cr> ok <0>
0000011   CONSTANT   OwnerR/W  <cr> ok <0>
0000011   CONSTANT   R/WMode  <cr> ok <0>
DECIMAL  <cr> ok <0>


\ create a data buffer in memory and fill it with 100 asterisks
VARIABLE   DataBuffer   96  VALLOT  <cr> ok <0>

DataBuffer   100   ASCII  *  FILL  <cr> ok <0>


\ create a file, ask for read/write access mode,
\ write the contents of the data buffer out to the file,
\ and close the file.
"  MyFile" 1+  OwnerR/W   R/WMode   $CREATE .  .  <cr> 0 3 ok <0>

DataBuffer  100  3  $WRITE  <cr>  0 ok <0>

3   $CLOSE  <cr> 0 ok <0>


\ now we will use the OS-9 'List' command from within MACH2 to
\ verify that the data was written to the file.
$   LIST   MyFile  <cr> .
.............................................................................
.....................
ok <0>
```

Other File Handling Words

OS-9 provides many other file handling words, several of which will soon be added to the list of MACH2/OS-9 file handling words. All of the OS-9 file handling routines are discussed on pages 15-1 through 15-26 in the OS-9/68000 Operating System Technical Manual.

Using the OS-9 File Calls from Assembly Language

The following code definition shows how the I$Read command could be used from assembly language:

```
CODE   $READ  (  buffer-address  len  path#  -  #bytesread  errorcode  )
    MOVE.L   (A5)+,D0      \ get path number
    MOVE.L   (A5)+,D1      \ get # bytes to read
    MOVEA.L (A5)+,A0       \ get address of buffer where data should be put
    OS9      I$Read        \ read the data
    BCS.S    @1            \ error if carry set
    MOVE.L   D1,-(A5)      \ # bytes actually read returned in D1
    CLR.L    -(A5)         \ return 0 error code
    RTS

@1CLR.L    -(A5)          \ error occurred, return 0 for #bytesread
    EXT.L    D1            \ extend the word-length error returned
    MOVE.L   D1,-(A5)      \ place errorcode on stack
    RTS
END-CODE
```

ERROR HANDLING

?OS9ERROR and ERRORPATH

The words ?OS9ERROR and ERRORPATH are two built-in OS-9 error handling tools
which are located in the OS-9 vocabulary. ?OS9ERROR expects to be passed an error
code returned by an OS-9 system call. If the error code indicates that an error has occurred,
?OS9ERROR will display the error number in OS-9 format (i.e. Error number #mmm.nnn).
If no error occurred, ?OS9ERROR will do nothing. Here is the definition of ?OS9ERROR:

```
CODE  PrintErr  (  n  -  )
    MOVE.L  (A5)+,D1
    MOVE.L  ERRORPATH(A6),D0
    OS9     F$PErr
    RTS
END-CODE

:  ?OS9ERROR  (  errorcode  -  )
   errorcode
   IF
        CR
        errorcode    PrintErr
        ABORT
   THEN  ;
```

The OS-9 system call F$PErr is the routine which actually evaluates the error code and
prints an error message if required. Note that ?OS9ERROR will consume the error code
passed to it. Here is an example of the use of ?OS9ERROR:

```
ALSO    OS-9
DECIMAL

255    ?OS9ERROR    <cr>
Error  #000:255

257    ?OS9ERROR    <cr>
Error  #001:001
```

Bits 0-7 in the value passed to F$PErr will be used for the error number on the right of the
colon. Bits 8-15 are used for the error number to the left of the colon. Error numbers
000:000 through 064:255 are reserved for the operating system.

If the path number of an open file is stored in ERRORPATH, ?OS9ERROR will search the file for the error message text which corresponds to the error code. The discussion of F$PErr on page 14-25 of the OS-9/68000 Operating System Technical Manual describes the format an error message file must have. The error message file which corresponds to the OS-9 system errors is called 'ErrMsg' and should be located in your 'SYS' directory. The following example shows how you may turn on 'long error message reporting' from within MACH2:

```
ONLY   FORTH   DEFINITIONS   <cr> ok <0>
ALSO   OS-9   <cr> ok <0>
DECIMAL   <cr> ok <0>

"   SYS/ErrMsg"   1+   3   $OPEN   .   .   <cr> 0 3 ok <0>

3   ERRORPATH   !   <cr> ok <0>

\   now, type some gibberish characters after the '$' word
\   to generate an error message.
$   ;AKDF;LKJADF   <cr>
Error #000:216   (E$PNNF)   File not found.
  The pathlist does not lead to any known file.
```

EXCEPTION HANDLING

Each OS-9 process (or task) may handle the basic 68000 exceptions privately if so desired. The following list shows the 68000 exception errors which may be handled privately by a task (and their corresponding offsets into the 68000 exception vector table):

Offset	Exception Error	Offset	Exception Error
$08	Bus Error	$1C	TRAPV Instruction
$0C	Address Error	$20	Privilege Violation
$10	Illegal Instruction	$28	Line 1010 Emulator
$14	Zero Divide	$2C	Line 1111 Emulator
$18	CHK Instruction		

Handling an Exception Error

When a custom exception error handling routine is called, the 68000 registers will contain the following information:

D7.w	Exception vector offset
A0	Program Counter (PC) value when exception occurred.
A1	Stack pointer (SP) value when exception occurred.
A5	User's register stack image (D0-D7/A0-A6) when exception occurred.
A6	User's primary global data pointer.

An exception vector is a memory location from which the 68000 processor will fetch the address of the routine which will handle that exception. Exception vectors are always located in a table which resides in low memory (from address $0000 to address $03FF). The contents of the 68000 exception vectors should never be altered directly. Always use the methods described in this section to set up an exception handling routine. The offset from the start of the exception vector table to the exception currently being processed will be passed to an exception handling routine in the lower word of the D7 register.

The A0 register will usually contain the value which was in the program counter when the exception occurred. The program counter value usually points to the next unexecuted instruction, however, for bus and address error, the program counter value is unpredictable (see page 40 of the Motorola 68000 Programmer's Reference Manual, 4th Edition).

All user register values (D0-D7.A0-A6) at the time of the exception will be stacked up on the A5 stack when the exception handler is called. The user's stack pointer value at the time of the exception will be passed to the exception handler in the A1 register. This gives the exception handling routine the user's complete register image at the time of the exception. The first action of a custom exception handling routine should be to restore the user's complete register image.

66

Exception Handling

Example Exception Handling Routines

Example exception error handling routines for an 'Address error' and a 'Line 1010 emulator error' are shown below. Note that for the Line 1010 error we are able to print out information about the PC value at the time of the exception. We cannot print out PC information for the Address error because the PC value information passed to the exception handling routine will not be valid for an Address error exception. Note that both routines restore the user's complete register set before proceeding:

```
\ ==========   Address Error   ==============================

: (address_error)   (  -  )
        CR
        ."    Address Error"
        CR
        ABORT   ;

CODE  address_error   (  -  )
        MOVEA.L     A1,A7               \ restore user stack pointer
        MOVEM.L     (A5),D0-D7/A0-A6 \ restore user registers
        (address_error)                 \ call higher-leve address error
END-CODE                                \ exception handling routine


\ ==========   Line 1010 Emulator   =========================

: (Line1010)   {  PC  |  oldbase   }   \ take value off stack and place
        CR                              \ in the PC named input parameter
        ."    Line 1010 error at   "    \ print message
        BASE  @  ->  oldbase            \ save current BASE
        HEX  PC  .                      \ set BASE to HEX, print PC value
        oldbase  BASE  !                \ restore previous BASE
        CR
        ABORT   ;

CODE   Line1010   (  -  )
        MOVEA.L     A1,A7               \ restore user's stack pointer
        MOVE.L      A0,-(A7)            \ save PC value on sytem stack
        MOVEM.L     (A5),D0-D7/A0-A6 \ restore user's registers
        MOVE.L      (A7)+,-(A5)         \ place saved PC value on param
        (Line1010)                      \ stack and pass to higher
END-CODE                                \ level exception handling routine
```

67

Exception Handling

Creating an Exception Table

In order to let OS-9 know about your task's custom exception handling routines, you must create an 'exception table (service request initialization table). Each exception error your task will handle in a custom manner should have an entry in the table. Each entry consists of 2 words (16 bits) of data. The first word should be the exception vector offset for the exception and the second word should contain the word-length offset to the custom exception handling routine for the exception. The end of the table must be marked with a word-length -1 value:

```
HEADER     ExcpTbl
      DC.W     $00C,Address_Error-*-2    \ offset to Address_Error routine
      DC.W     $028,Line1010-*-2         \ offset to Line1010 routine
      DC.W     -1                        \ end of table
```

NOTE: In the Exception Table example found on page 14-39 of the OS-9/68000 *Operating System Technical Manual* a 4, *rather than the 2 shown above, is subtracted* in the calculation of the word-length offset to the exception handling routine. This discrepancy is due to current deviations between the MACH2 assembler and the OS-9 assembler in the functioning of the "*" assembler word.

Installing the Exception Table

Once the custom exception handling routines have been written and the exception table has been created, the OS-9 user mode system call F$STrap must be used to install (let OS-9 know about) the custom exception error handling routines:

```
CODE    Install    ( - f )
        LEA        ExcpTbl,A1      \ pass the exception table address in A1
        MOVEQ.L    #0,D0
        MOVEA.L    D0,A0           \ use current stack if exception occurs
        OS9        F$STrap          \ calling F$STrap
        BCS.S      @1              \ if carry bit set, error occurred, return code
        MOVEQ.L    #0,D1           \ no error occurred, return 0 error code
@1      EXT.L      D1              \ extend error code, if any
        MOVE.L     D1,-(A5)        \ place error code on stack
        RTS
END-CODE
```

For more information on custom exception error handling refer to page 14-39 of the OS-9/68000 *Operating System Technical Manual*.

INTER-PROCESS COMMUNICATION: SIGNALS

OS-9 processes (tasks) may communicate with each other by passing signal codes.
A signal code is a word-length (16-bit) value. Four signal code values have predefined
meanings:

Symbol	Value	Signal Meaning
S$Kill	0	System abort (unconditional)
S$Wake	1	Wake up process
S$Abort	2	Keyboard abort
S$Intrpt	3	Keyboard interrupt
	256-65535	User defined

An Example Signal Intercept Routine

The 'Vector_Signal' routine shown below is the signal intercept routine used by MACH2.
When this signal intercept routine is called, it will be passed the word-length signal code in
the D1 register and the A6 register will hold the address of MACH2's program data area.
Normally, an intercept routine is terminated with the F$RTE system call. However,
according to the discussion on F$Icpt (see page 14-20 of the OS-9/68000 OS Technical
Manual), the 'MOVEM.L' and 'RTR' instructions may be substituted as a faster alternative.

```
CODE   Vector_Signal   (  -  )
    EXT.L    D1                      \ signal code is in D1.W
    MOVE.L   D1,-(A5)                \ place code on MACH2 stack
    MOVE.L   RESPONSE(A6),A0         \ get address of MACH2 signal vector
    JSR      (A0)                    \ execute signal routine
    MOVEM.L  (A7)+,D0-D7/A0-A7       \ restore registers
    RTR                              \ continue mainline execution
END-CODE
```

'Vector_Signal' is the assembly language interface to signal reception in MACH2.
'Vector_Signal' places the signal code received on the MACH2 parameter stack so that a
higher level FORTH routine may be used to respond to the signal. After the higher level
routine has finished execution, 'Vector_Signal' takes care of 'cleaning up' after the signal.

RESPONSE

RESPONSE is a MACH2 system variable found in the OS-9 vocabulary. The RESPONSE
variable is used to hold the address of the higher level FORTH routine to be used to
respond to signal receptions. Initially, RESPONSE holds the address of the 'Handle_Signal'
routine. 'Handle_Signal' is the default routine used by MACH2 to respond to the four system
defined signals listed above. The 'Handle_Signal' routine is shown later in this section.

69

Signals

Installing a Signal Intercept Routine

The F$Icpt user mode system call is used to tell OS-9 where the current process' signal intercept routine is located. It is important to note that if a process receives a signal and it does not have a signal intercept routine installed, the process will be aborted. The 'Vector_Signal' signal intercept routine discussed previously is automatically installed each time MACH2, or a TURNKEY application (an executable module) created by MACH2, starts up. The CODE definition below shows the signal intercept installation routine used to install the 'Vector_Signal' routine:

```
CODE   Install_Icpt   (  -  )
    LEA      Vector_Signal,A0   \ pass address of signal intercept
    OS9      F$Icpt             \ routine in A0
    RTS
END-CODE
```

F$Icpt is passed the address of the 'Vector_Signal' routine in the A0 register and the address of the current program's data area in the A6 register. The current program's data area address is already in the A6 register when 'Install_Icpt' is run so the set up of the A6 register is not explicitly shown.

MACH2's High Level FORTH Signal Handling Routine

'Handle_Signal' is the FORTH routine used by MACH2 to respond to the four system-defined signals:

```
:   Keyboard_Interrupt  (  -  )        \ This routine handles
    CR                                  \ CONTROL C keyboard
    ."   Keyboard Interrupt    "        \ interrupts.
    CR
    ABORT   ;
```

```
\ Standard_Signal handles signals with signal codes other
\ than 0, 1, 2, 3 by printing out a message followed by the
\ signal code itself.
```

```
:   Standard_Signal  {  num  |  oldbase  -  }
    BASE   @   ->  oldbase            \ save the current base
    DECIMAL                           \ set the base to DECIMAL
    CR
    ."   Signal received:  #  "   num   . \ print the signal number received
    oldbase  BASE  !  ;               \ restore the old base
```

70

Signals

```
:  Handle_Signal   (  id  -  )
    id
    CASE
          0  OF  BYE                    ENDOF  \ handle system abort
          1  OF                         ENDOF  \ this signal is never received
          2  OF  BYE                    ENDOF  \ handle keyboard abort
          3  OF  Keyboard_Interrupt     ENDOF  \ handle keyboard interrupt
          id  Standard_Signal                  \ handle user defined signal
    ENDCASE  ;

:  Intercept   (  -  )
    [']  Handle_Signal   RESPONSE  !         \ Signal handling is vectored
                                             \ through the variable
    Install_Icpt   ;                         \ 'RESPONSE'.
```

'Intercept' is a high level version of the routine MACH2 runs upon start up to set up its own signal intercept handling.

A Custom Signal Handling Routine

The code examples below show how a custom signal handling routine could be written and installed. Note that once this custom routine is installed, it will supersede the MACH2 signal handling routine used to respond to system defined signals (system abort, keyboard interrupt, and keyboard abort). Normally, a custom signal handling routine should also handle the four system defined signals:

```
ONLY   FORTH   DEFINITIONS
ALSO   OS-9
DECIMAL

:  Catch_Signal   (  code  -  )
    CR
    ."  Signal  "
    code
    CASE
          300  OF  ."  three hundred" ENDOF
          400  OF  ."  four hundred" ENDOF
          500  OF  ."  five hundred" ENDOF
          600  OF  ."  six hundred " ENDOF
          DROP."  unknown"
    ENDCASE  ;

:  Install_SHandler   (  -  )
    [']  Catch_Signal   RESPONSE  !  ;
```

71

Signals

Sending Signals

To send a signal use the F$Send user mode system call (see page 14-29 of the OS-9/68000 OS Technical Manual). F$Send expects to be passed the intended receiver's process ID and the signal code to send. The following example shows how a process can send a signal to itself. Before a process may send a signal to itself it must use the 'FetchID' routine to find its process ID number:

```
CODE   FetchID   ( - n )
    OS9         F$ID          \ get the caller's process ID
    BCS.S       @1            \ if error, go to @1
    EXT.L       D0            \ extend the word-length process ID
    MOVE.L      D0,-(A5)      \ put process ID on parameter stack
    MOVEQ.L     #0,D1         \ put a 0 (= no error) in D1
@1 EXT.L        D1            \ if an error was returned, extend it
    MOVE.L      D1,-(A5)      \ and place it on the stack
    ?OS9ERROR                 \ ?OS9ERROR will take the error code
    RTS                       \ off the stack, examine it, and abort
END-CODE                      \ if non-zero (see error handling section)


CODE   Send   ( n - )
    FetchID                   \ get the process ID for this process
    MOVE.L      (A5)+,D0      \ take the process ID off the stack
    MOVE.L      (A5)+,D1      \ take the signal code off the stack
    OS9         F$Send        \ send the signal code to this process
    BCS.S       @1            \ if error, go to @1
    MOVEQ.L     #0,D1         \ assume no error (=0)
@1 EXT.L        D1            \ extend error code, if any
    MOVE.L      D1,-(A5)      \ place error code on stack
    ?OS9ERROR                 \ and pass it to ?OS9ERROR for examination
    RTS
END-CODE
```

Now, after 'Install_SHandler' is used to install our custom signal handling routine, we can try sending a signal to our process:

```
Install_SHandler   <cr>      \ install custom signal handling routine

400   Send   <cr>            \ send signal # 400 to our process
Signal four hundred  ok <0>

5   Send   <cr>              \ send signal # 5 to our process
Signal unknown  ok <0>
```

72

Signals

PROCESS PARAMETER PASSING

Each time a new process is created in OS-9 (for example, by typing the name of an executable module from the shell) a string of parameters may be passed to the process. For example, when MACH2 is started up you have the option of passing an additional parameter to MACH2 which specifies the memory size which should be allocated for the user's code space:

$ MACH2   -$45000

MACH2's first action upon start up is to parse the parameter string, if any, and look for valid input parameters.

PARAM_PTR

The MACH2 word PARAM_PTR, located in the OS-9 vocabulary, is included so that user's may incorporate process parameter passing into their own executable (TURNKEY) modules. PARAM_PTR will return the address of the null-terminated parameter input string passed to a process when the process was started.

A Parameter Passing Example

The example on the following pages shows how to create a simple TURNKEY application (an executable module) called 'Parser' which expects to be passed one numerical input (decimal or hexadecimal). The numerical input should be immediately preceded by a hyphen. The first action performed by 'Scan', the highest level word in the program, is to use PARAM_PTR to get the address of the parameter string passed to 'Parser'. 'Scan' passes the address of the input parameter string to 'Analyze' for parsing. 'Analyze' will return a flag, indicating whether a valid input was found, and the input value. If a valid input was found 'Scan' will print the value out and terminate by exiting to the OS-9 shell. Otherwise, 'Scan' will print out an error message and exit to the OS-9 shell.

Additional Information

For more technical information on process parameter passing see the discussion of F$Fork on pages 14-14 and 14-15 of the OS-9/68000 Operating System Technical Manual. For more discussion on the format of an OS-9 command line see the discussion on pages 5-2 and 5-3 of the OS-9 Operating System User's Manual.

\ ================= Parameter Parser =================

ONLY   FORTH   DEFINITIONS
ALSO   OS-9

DECIMAL

\ 'Analyze' will return a flag indicating whether the input string contains a valid
\ input, i.e. a hypen followed by a valid number. A '$' character may be used
\ to indicate a hexadecimal number.
\ 'Analyze' will return a true (-1) flag on top of the stack if a valid input is found.
\ A false (0) flag will be returned otherwise. The second number on the stack
\ will either be the number, if valid, or a zero.

```
:  Analyze  {  addr  |  char  -  n  f  }
            \ == strip leading spaces ==
         BEGIN
            addr   C@   ->   char
            1   +>   addr
            char   ASCII   -   =          \ search until either a '-' or a
            char   0=                     \ null character is found
            OR
         UNTIL

         char
         IF
            \ == check for the '-$4500' case
            addr   C@   ASCII   $   =
            IF
                  HEX
            ELSE
                  DECIMAL
                  -1   +>   addr           \ back up addr to point to hyphen
            THEN

            \ addr must point one character before the numeric portion of
            \ the string.  Examples:
            \ -$340          -1234
            \  ^              ^
            addr   NUMBER?                 \ NUMBER? leaves flag and value
                                           \ see NUMBER? glossary page.
         ELSE
            0  0                           \ no parameter string, leave false
         THEN  ;                           \ flag and zero value.
```

74

Process Parameter Passing

```
\ 'Scan' is an example utility that may be used to analyze the parameter string
\ passed to an OS-9 executable module.

:  Scan   {  |  flag   num  }
        CR
        PARAM_PTR     Analyze
        ->   flag
        ->   num

        flag
        IF
              DECIMAL
              ."   Parameter data  :   "   num   .
        ELSE
              ."   No input data.  "
        THEN
        CR
        BYE   ;

CR
.(   TURNKEY   Scan   Parser   )
```

Executing the 'Parser' Example Application Module

After you have loaded the 'Parser' example program into MACH2 and used TURNKEY
to create the executable 'Parser' module, 'Parser' could be used from the OS-9 Shell
as follows:

```
$    Parser   -1234
Parameter data  :  1234

$    Parser   -$2000
Parameter data  :  4096

$    Parser
No input data.

$
```

Process Parameter Passing

OS-9 TRAP MODULES

Trap modules (trap handlers) are independent code modules which may be created
by any language that compiles to machine code (FORTH, assembly, C). MACH2 has
facilities for the creation of two types of trap modules: 'MACH' format trap modules and
'generic' trap modules. 'MACH' format modules are designed to be 'called' by MACH2
or a program created using MACH2. 'Generic' trap modules may be called by a program
written in any language.

Why Use Trap Modules ?

One reason is that trap modules allow sections of infrequently used code (device
initialization routines, etc.) or general purpose code (a set of functionally related routines
which may be used by many programs) to be removed from the main program to reduce
the amount of execution memory required by the program. When the code in the trap
module is needed, the main program can 'call' the trap module. The trap module will be
loaded into memory, if necessary, and executed. When the trap module has finished
execution the main program can 'unload' the trap module (remove it from memory).

Another reason is that the use of trap modules allows for parallel program development.
One programmer can work on the main program while one or more other programmers
work on trap module code.Since trap modules are independent code modules, they
may be independently and individually tested and executed.

MACH2's 'generic' format trap modules allow for parallel program development AND
multi-language program development ! For example, in a large industrial control project,
the control engineers could use FORTRAN to develop their control algorithms and the
hardware engineers could use FORTH to bring up the system hardware and to write
and test the required device driver trap modules. The development of the device
drivers would not be held up by the development of the main program or vice versa
and the control engineers could perform system integration tests with the device
driver trap modules at their own convenience.

Organization of This Section

The first part of this trap module discussion centers on the creation and use of 'MACH'
format trap modules. The code for an example 'MACH' format trap module is presented
and discussed. The second part of this section describes the creation and use of
'generic' format trap modules. An extensive example which demonstrates how a
'generic' trap module may be called from an OS-9 C program is presented. The MACH2
program listing for the 'generic' trap module and the OS-9 C listing for the 'main program'
which calls the trap module are both included.

Many assembly language examples have also been provided for those programmers who
wish to gain an in-depth understanding of the OS-9 user trap handler mechanism.

'MACH' FORMAT TRAP MODULES

A 'MACH' format trap module is a trap module which may be called only from within MACH2 or by an executable module (TURNKEY application) created by MACH2. The reason for this requirement is that during execution, a 'MACH' format trap module will assume that the A6 register points to a valid 'MACH2' data area. Since many MACH2 kernel words are accessed via a jump table located in MACH2's data area, and many MACH2 words reference system variables located in the MACH2 variable space (which. is also located in the MACH2 data area), this assumption allows a 'MACH' format trap module to use any word in the MACH2 kernel. A 'MACH' trap module also assumes that it may use the parameter stack to pass parameters between itself and the calling program.

Creating a 'MACH' FormatTrap Module

To create a 'MACH' format trap module, use the word 'MACHMODULE' :

     MACHMODULE   **<main word>**   **<module name>**

MACHMODULE should be used after the code to be placed in the module has been loaded into memory. <main word> is the word which will be run when the trap module is later accessed. <module name> is the name for the module.

When MACHMODULE is executed, MACH2 takes all code in the user's code area, appends some initialization code, and writes it all out to a new trap module with the given name. After MACHMODULE has completed execution, it will exit MACH2 and return to the OS-9 shell. The new trap module will be located in the current execution directory.

The <main word>

The <main word> in the trap module will be called via a 'JSR' instruction when the trap module is executed (described in more detail later in this section). A single selector value will be passed to the <main word> on the parameter stack. The <main word> may or may not return parameters to the calling program on the parameter stack. Thus, the <main word> should have at least the following stack notation:

     <main word>   **(selector -**   **)**

An Example 'MACH' Format Trap Module

The code for an example 'MACH' format trap module is shown below. This simple program will analyze the selector value passed to it and print the corresponding string before returning to the calling program. After the program has been loaded, MACHMODULE is used to turn the code into a trap module named 'NumModule':

```
3   CONSTANT   Three
2   CONSTANT   Two
1   CONSTANT   One

:   DoThree  ( - )   ." Three " CR  ;
:   DoTwo    ( - )   ." Two " CR  ;
:   DoOne    ( - )   ." One " CR  ;

\ PrintNums is the <main word> for this trap module.
:   PrintNums  ( selector - )
     CASE
           One   OF   DoOne  ENDOF
           Two   OF   DoTwo  ENDOF
           Three OF   DoThree ENDOF
     ENDCASE  ;

\ Now we will create the trap module

MACHMODULE   PrintNums   NumModule
```

Assigning a Trap Module to a Trap Vector

The code in trap modules is accessed throught one of 16 software trap exception vectors provided by the 68000 microprocessor. The table shows which trap vectors are reserved and which trap vectors are available for use by trap modules:

Vector Number	Use
0	Used by OS-9 for system calls.
1-12	Available for trap module use.
13	Used by 'C' for I/O. Used by MACH2 for its disassembler/debugger.
14, 15	Used by the OS-9 math packages.

To access a trap module from within MACH2 you must first let MACH2, and OS-9, know which software exception vector should be used to access the module, you must 'assign' a trap vector number to the trap module:

```
5   CONSTANT   NumTrap

"   NumModule"   1+   NumTrap   ASSIGNMODULE
```

78

'MACH' Format Trap Modules

ASSIGNMODULE is the MACH2 word used to assign software exception vectors to trap modules. ASSIGNMODULE expects to be passed the address of a null-terminated string which contains the module name and the number of the vector you wish to assign to the trap module.

The word ' " ' lays a string into memory which has both a leading count byte and a trailing null byte. Since ' " ' returns the address of the count byte, a ' 1+ ' is used to index over the count byte so that the address points directly at the start of a null-terminated string (see the ' " ' glossary page).

In the example, software exception vector #5 was assigned to the newly created trap module. The CONSTANT NumTrap was used for readability.

Calling the Trap Module

Now that a trap module has been created and assigned a software vector, the module may be called:

```
:  OneString   (  -  )    TCALL   NumTrap,1  ;
:  TwoString   (  -  )    TCALL   NumTrap,2  ;
:  ThreeString  (  -  )    TCALL   NumTrap,3  ;

\  Now let's call the module ...

OneString   <cr> One
ok <0>
```

The MACH2 word 'TCALL' is used to call trap modules. TCALL may be compiled or used interactively for testing purposes. TCALL is used in the following format:

> **TCALL**   **<vector#>,<selector>**

The word 'OneString' was used above to call the trap module assigned to vector #5 (our 'NumModule' trap module) with a selector of 1 (which caused NumModule to print out the "One" string). The selector is the number which is passed to the trap module on top of the stack. The selector value must be in the range ±32767.

Note that TCALL is actually a MACH2 assembler word. The definition 'OneString' above could also have been written in assembly language:

```
CODE  OneString  (  -  )
   TCALL   NumTrap,1
   RTS
END-CODE
```

79

'MACH' Format Trap Modules

Low-level 'MACH' Format Trap Module Information

The following information on 'MACH' format trap modules is for those who wish to learn how 'MACH' format trap modules are implemented at the OS-9 system call level.

'MACH' Format Trap Initialization Code

When MACH2 creates a 'MACH' trap module, it installs the following initialization code:

```
: mainword ( n - ) ;                          ▲ module code
    ︵‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾︵︵︵
    CODE  InitModule  ( - )                   ▼ initialization code
        MOVEA.L  (A7),A6          \ Line 1
        MOVEQ.L  #0,D0            \ Line 2
        MOVE.W   4(A7),D0         \ Line 3
        MOVE.L   D0,-(A5)         \ Line 4
        JSR      mainword         \ Line 5
        MOVEM.L  (A7)+,A6-A7      \ Line 6
        RTS
    END-CODE
```

When a trap handler is called, the system stack contains the following information:

+8	caller's return PC	(4 bytes)
+6	vector number	(2 bytes)
+4	selector	(2 bytes)
(A7) → +0	caller's A6 register	(4 bytes)

The instruction in Line 3 above indexes 4 bytes into the stack frame to get the 2-byte selector passed to the trap module by the calling program. In Line 4 the selector is placed on the parameter stack (MACH2 uses the A5 register to maintain its parameter stack) and then in Line 5, a 'JSR' to the <main word> in the trap module is performed.

The instruction in Line 6 is used often in the OS-9 examples to reset the A6 register, reclaim stack space (the selector and vector number space), and set up the A7 register (to point at the caller's return PC) in one operation. The action relies heavily upon an internal characteristic of the 'MOVEM' instruction. The source addressing mode overrides the destination mode in register assignment. The instruction 'MOVEM.L (A7)+,A6-A7' will leave the A7 register pointing at the 'caller's return PC', regardless of the data moved into the A7 register.

Assembly Language Definition of ASSIGNMODULE

The assembly language definition of ASSIGNMODULE is shown below. MACH2 keeps an internal table, named (for demonstration purposes) 'MODULE_TABLE', of 16 4-byte locations in memory. The sixteen locations correspond to the software exception vectors 0-15. Each location contains either a 0 (if the corresponding vector is unassigned) or the address of the name string for the trap module assigned to the vector.

```
    \  Allocate variable space for 16 long addresses

    DECIMAL

    VARIABLE    MODULE_TABLE
    15   4  *  VALLOT

    CODE  ASSIGNMODULE  ( a n -  )
        MOVE.L   (A5)+,D0
        ANDI.W   #$F,D0
        ASL.W    #2,D0                        \ 4 *
        LEA      MODULE_TABLE(A6),A0
        MOVE.L   (A5)+,0(A0,D0.W)             \ store address
        RTS
    END-CODE
```

You will see in the 'Reading a Trap Module into Memory' discussion below that if MACH2 is asked to call a module using a vector which does not have a valid name string address in the MODULE_TABLE that an error condition will occur.

Reading a Trap Module Into Memory

The first time a trap module is accessed, special initialization actions must be performed. The assembly language routine shown on the following page is used by MACH2 to handle the first-time access of a trap module. This routine is based on the OS-9 example found on page 11-4 of the OS-9/68000 Operating System Technical Manual.

The first action performed by TRAPINIT is to extract the vector number from the stack frame of information passed on the system stack and to use the vector number to index into MACH2's MODULE_TABLE to find the trap module name which has been assigned to the vector (with the use of ASSIGNMODULE, as described earlier). If a name has not been assigned to the vector, MACH2 will issue an error message.

'MACH' Format Trap Modules

Next, the OS-9 routine F$TLink (page 14-44 in the OS-9/68000 OS Technical Manual) is used to read the trap module into memory (if it has not already been read into memory by another program) and to initialize static storage for the trap handler if required. If a module with the specified name is not found MACH2 will display a '<module name> missing' error message.

At this point the module has been properly initialized. The program counter is backed up so that it points once again to the original 'TCALL' instruction. The 'TCALL' instruction is re-run and, since the module is now available in memory, the trap module--NOT the TRAPINIT code--is executed.

```
\ Installs trap handler, and then executes the first trap call.

CODE   TRAPINIT  (  -  )
       MOVEM.L    D0-D1/A0-A2,-(A7)      \ save registers used
       MOVE.W     26(A7),D0              \ fetch vector ID
       SUBI.W     #$80,D0                \ remove TRAP #0 offset
       LEA        MODULE_TABLE(A6),A0
       MOVEA.L    0(A0,D0.W),A0          \ string address
       TST.B      (A0)                   \ check for null string
       BEQ        @BadTrap

       MOVE.L     A0,-(A7)               \ save name in case of error
       LSR.W      #2,D0                  \ User Trap Number
       MOVEQ.L    #0,D1                  \ no optional memory override
       OS9        F$TLink                \ read module into memory
       BCS.S      @ModuleMissing

       ADDQ.L     #4,A7                  \ throw away module name
       MOVEM.L    (A7)+,D0-D1/A0-A2      \ restore registers
       ADDQ.L     #8,A7                  \ discard excess stack info

       SUBQ.L     #4,(A7)                \ back up over trap instruction
       RTS                               \ and selector

\ A vector has not been assigned to this module.
@BadTrap                                 \ error handling not shown

\ The specified module name was not found.
@ModuleMissing                           \ error handling not shown
END-CODE
```

82

'MACH' Format Trap Modules

'GENERIC' FORMAT TRAP MODULES

A 'generic' format trap module is a trap module created by MACH2 which may be called by a program written in any language. A 'generic' format trap module may be called from other languages because it does not make any assumptions about the register usage or parameter passing techniques of the calling program. Since a 'generic' trap module cannot assume that the A6 register points to a valid MACH2 data area, it can only use a subset of the available MACH2 kernel words. During execution, a 'generic' format trap module can make full use of the MACH2 stacks (an initialization routine will set them up for the module) but the stacks cannot be used to pass parameters back to the calling program.

Words Which May Be Used in a 'Generic' Format Trap Module

The lists below contain all of the MACH2 words which may be compiled into a 'generic' format trap module program. All of the program control structure words, all of the words relating to local variables, and most of the arithmetic and stack manipulation words are included in the list. These types of words are either immediate compiling words which generate machine code when they are compiled (IF...THEN , ['] , etc.), or they are MACH words whose code is laid in line during compilation (C@ , 0= , -> , etc.). These words are acceptable for use in a 'generic' trap handler module program because they do not use internal MACH2 system variables(variables are A6 dependent) and they do not generate MACH2 kernel references (the jump table is A6 dependent). All MACH2 assembler words may also be used. Note that the MACH2 floating point words and the MACH2/OS-9 file words are not included in the list because they would generate jump table references when compiled.

!	2*	?DUP	ELSE	MAX	W!
"	2+	@	ENDCASE	MIN	W@
+	2-	ABS	ENDOF	NEGATE	WHILE
+!	2/	AGAIN	EXECUTE	NOT	W_EXT
+>	2DROP	AND	EXIT	OF	XOR
+LOOP	2DUP	BEGIN	I	OR	[']
-	2OVER	BYE	I'	OVER	^
->	2SWAP	C!	IF	R>R@	(
0<	<	C@	J	REPEAT	
0=	<>	CASE	LEAVE	SWAP	
0>	=	DO	LITERAL	TCALL	
1+	>	DROP	LOOP	THEN	
1-	>R	DUP	L_EXT	UNTIL	

83

Generic Format Trap Modules

A MAKEMODULE Utility Command

VERBOSE is a MACH2 system variable used to control compilation error messages. When the VERBOSE variable contains a negative value, any compiled references to words which are not allowed in a 'generic' trap module will be flagged during loading:

        -1   VERBOSE   !   <cr>

        : Test (-) CONVERT ; <cr> CONVERT may not be used by MAKEMODULE.

Since VERBOSE was holding a negative value, the compiler was watching out for compiled references to words which are invalid in a 'generic' trap module. Note that although CONVERT was flagged as being an invalid reference, the loading process was not aborted.

Creating a Generic Trap Module

The process of creating a generic MACH2 trap module is very similar to the process used to create a MACH2 format MACH2 trap module. The only difference is that the word MAKEMODULE (instead of MACHMODULE) is used to create generic MACH2 trap modules:

        MAKEMODULE   <mainword>   <modulename>

MAKEMODULE should be used after the program to be run by the generic trap module has been loaded into the MACH2 environment. <mainword> is the word which will be executed when the trap module is 'called'. <modulename> is the name OS-9 will use to identify the trap module. When MAKEMODULE is executed it will create a generic trap module, append some initialization code, and exit to the OS-9 shell.

'Generic' Format Trap Modules

An Example 'Generic' Trap Module

A listing of a program to be turned into a 'generic' trap module is shown below. The program creates a device-specific driver module that simulates the polling of three analog-to-digital devices. Since the FORTH language is especially suited for machine and device control, the MACH2 system should be especially useful for the creation of low-level, generic, OS-9 device driver modules. On the other hand, the MACH2 system might not be as suitable for the creation of generic I/O trap modules since none of the MACH2 I/O words are allowed in a 'generic' trap module.

```
\ Program listing for generic trap module.

1  CONSTANT   A/D1                        \ selector value for device 'A/D1'
2  CONSTANT   A/D2                        \ selector value for device 'A/D2'
3  CONSTANT   A/D3                        \ selector value for device 'A/D3'

\ These words simulate the responses of the A/D devices.
\ For demonstration purposes, constant values are returned.

:  ReadA/D1  ( - n )   123   ;            \ device 'A/D1' always returns 123
:  ReadA/D2  ( - n )   456   ;            \ device 'A/D2' always returns 456
:  ReadA/D3  ( - n )   789   ;            \ device 'A/D3' always returns 789

\ 'Main' is the main word in this trap module.  This trap module is simulating a driver
\ which takes reading from one of three analog-to-digital devices.  The value read
\ is returned in the D0 register.

:  Main  {  frame  selector  |  data  -  }
        selector                                      \ check the selector...
        CASE
              A/D0F    ReadA/D1   ->   data ENDOF  \ and take a reading.
              A/D0F    ReadA/D2   ->   data ENDOF
              A/D0F    ReadA/D3   ->   data ENDOF
              0  ->  data
        ENDCASE
        data   frame   !   ;                           \ save the data into D0 (the D0 register
                                                       \ is the 'top' one in the stack frame.

\ Now this file may be loaded into MACH2.  After the loading process has completed, use
\ MAKEMODULE to create the trap module which will contain this A/D simulation code:

MAKEMODULE   Main   ADSampler   <cr>
```

Passing Parameters to a 'Generic' Trap Module

Parameter passing is the most important aspect of this program. The <mainword> in a 'generic' trap handler module should have the follwing stack notation:

    <mainword> (stackframepointer selector -)

When a 'generic' trap handler module is called, the top item on the stack will be the selector value passed in from the calling program. The second stack item will be the address of a 64 byte stack frame which holds the complete register image of the calling program at the time the trap handler module call was made. The stack frame address is provided so that if the calling program passes additional parameters to the trap module in registers, the trap module will be able to access the register contents.

The layout of the register stack frame is shown in the diagram below. If an additional parameter were passed in the D4 register (whose contents are located at an offset of 16 bytes into the stack frame area), the 'generic' trap module could retrieve the parameter value by inserting the sequence 'frame 16 + @' in the <main word> definition.

The 'Generic'	base A7	
Trap Module	+72 caller's return PC	
Execution Environment	+70 vector number	10
	+68 function code	bytes
	initial A7 +64 caller's A6 register	
	+60 A7	
	+56 A6	
	+52 A5	
	+48 A4	
	+44 A3	
	+40 A2	
	+36 A1	
	+32 A0	64
	+28 D7	bytes
	+24 D6	
	+20 D5	
	+16 D4	
	+12 D3	
	+8 D2	
	+4 D1	
" frame "— starting A7	+0 D0	
	▼	2000 bytes
starting A5	▼	400 bytes
starting A3	▲	
starting D7	▼	

86

'Generic' Format Trap Modules

Returning Results from a 'Generic' Trap Handler Module

The stack frame address also provides the 'generic' trap handler module with a flexible means of returning results to the calling program. For example, programs written in the OS-9 C Compiler expect single function results to be returned in the D0 register (see page 3-4 of the OS-9/68000 C Compiler User's Manual). In the example listing, the 'generic' trap module returns its result in the 'D0' field, which is located at an offset of 0 from the start of the stack frame:

        data   frame   (0 +) !

The 'Generic' Trap Module InitializationSequence

When MAKEMODULE creates a 'generic' trap module it appends initialization the initialization code, listed below, to the module. This routine performs three actions:

1. The complete register image of the calling program is saved on the system (A7) stack.

2. The four important MACH2 stacks (subroutine, parameter, loop, and floating point) are set up.

3. The trap selector value and a pointer to the register stack frame are placed on the parameter stack.

After the 'generic' trap module has completed execution, the saved stack image is restored. It is important to note that the initialization routine assumes that the 'generic' trap handler module has 3-4K bytes of available system stack space.

```
: MainWord      ( n - )   ;                    module code

CODE  InitGModule  ( - )                        initialization code
       MOVEM.L  D0-D7/A0-A7,-(A7)  \ save the calling program's registers
       LEA       -2000(A7),A5       \ allocate memory for the subroutine stack
       LEA       -400(A5),A3        \ allocate memory for the parameter/loop stacks
       MOVE.L   A3,D7              \ set up the floating point stack
       SUBI.L   #16,D7             \ allocate memory for the floating point stack
       MOVE.L   A7,-(A5)           \ put ptr to the register stack frame on param stack
       MOVE.W   4*17(A7),-(A5)     \ get selector value and place on parameter stack
       CLR.W    -(A5)              \ clear upper word
       JSR      <mainword>         \ call the main word in the generic trap module
       MOVEM.L  (A7)+,D0-D7/A0-A7  \ restore the calling program's registers
       MOVEM.L  (A7)+,A6-A7        \ remove params passed to trap handler by OS-9
       RTS                          \ return to calling program
END-CODE
```

87

'Generic' Format Trap Modules

The diagram on the previous page (not to scale) shows how the system stack is affected by the initialization code listed above. Whenever a trap handler module is called, the stack pointer will be in the 'initial A7' position. The 10 bytes of data at the top of the diagram are always passed to a trap module.

The first step in the initialization process involves pushing the complete register stack image onto the system stack. The register stack image takes up 64 bytes of stack space. At this point the stack pointer is left pointing directly at the last register in the stack image, the D0 register.

Next, the MACH2 stacks are set up. The parameter stack (A5) is set up to start 2000 bytes below the subroutine (A7) stack. The loop (A3) stack is set up to start 400 bytes below the start of the parameter stack. The floating point stack (D7) starts 16 bytes below the loop stack and grows downward.

After the MACH2 stacks have been set up, the initialization code indexes 68 bytes from the 'starting A7' position to get the word length selector value. This value, and the current value of the system stack pointer (points right at the saved value of the D0 register), are placed on the parameter stack so that they will get passed to the <mainword> in the 'generic' trap module.

When <mainword> has finished execution, the final two instructions in 'InitGModule' remove the register image and the initial parameters from the stack. This ensures that the system stack pointer is in the proper 'base A7' position (pointing right at the 'caller's return PC') when the 'RTS' instruction is executed.

Calling a 'Generic' Trap Module from C

On the following page is a listing of a C program, named 'trapcall.c', that demonstrates how a MACH2-generated 'generic' trap handler module may be called from C. This program, which was written using the OS-9/68000 C Compiler, is tuned to the register usage and parameter passing techniques used by the OS-9 C Compiler (see pages 3-3 through 3-5 of the OS-9/68000 C Compiler User's Manual).

The first two lines in the program are 'include' statements. The <stdio.h> file contains some standard C I/O definitions. The most important of these definitions is the 'printf' function which allows C programs to print messages. The <traps.c> file contains two C routines which allow C programs to access OS-9 trap handler modules:

    tlink()   Assigns/unassigns a trap handler module to a software trap exception vector.

    ttcall()   Calls a trap handler module.

'Generic' Format Trap Modules

```c
/* FILE:  trapcall.c  Written in the OS-9/68000 C Compiler.
          Example C program which calls the 'ADSampler' module
          written in MACH2 and created using MAKEMODULE.
*/

#include    <stdio.h>           /* standard C I/O definitions        */
#include    <traps.c>           /* contains the trap handler module access
                                   functions provided with MACH2     */

#define     ADTrap   5          /*  software exception vector used to access
#define     ModName  "ADSampler"  /* the A/D trap handler module       */
#define     AD1      1          /* name of the trap handler module     */
#define     AD2      2          /* these are the three possible selectors */
#define     AD3      3          /* which may be passed to the A/D module

Main()

{

        int      a1,a2,a3;

        printf("Beginning  C  to  MACH2  trap  module  linkage  example...\n\n");

        printf("Linking  trap  module  %s ...\n\n",ModName);
        TLink(ADTrap,0,ModName);

        printf("Reading  samples...\n\n");

        a1=Sample(AD1);
        a2=Sample(AD2);
        a3=Sample(AD3);

        printf("Simulated  Analog/Digital  devices  read:\n");
        printf("AD1=%d     AD2=%d     AD3=%d\n\n",a1,a2,a3);

        printf("Freeing  trap  number  %d\n",A/DTrap);
        TLink(ADTrap,0,0);

        printf("End  of  'TrapCall'  example'.\n");

}

Sample(device_num)
int    device_num
{
        return(TTCall(ADTrap,device_num));
}
```

'Generic' Format Trap Modules

The definitions of these routines will be discussed later. Five constants used in the program are defined next. 'ADTrap' is the number of the software exception vector which will be used to access the trap handler module. The 'tlink' routine will be used to perform this assignment. 'ModName' is a string constant which contains the name of the trap handler module to be called. In this example, the 'ADSampler' trap handler module will be called. 'AD1', 'AD2', and 'AD3' are three selector values which will be passed to the trap handler module when it is called.

The first action of the 'Main' routine in the program is to declare the three integer variables, 'a1', 'a2', and 'a3', which will receive the results returned by the trap handler module. Next, the 'tlink' routine is used to let OS-9 know that the program wishes to use software exception vector #5 to access the 'ADSampler' trap handler module. Now the program can call the trap handler module. The 'Sampler' routine, which uses the 'ttcall' routine to call the trap handler module (through software exception vector #5) and to pass the module a parameter, is called three times with three different selector values. The three values returned by the trap handler module are stored in 'a1', 'a2', and 'a3' and printed out. 'tlink' is then used once more, this time to 'unassign' the 'ADSampler' trap module to software exception vector #5.

The 'tlink' Routine

The 'tlink' routine is a 'glue' routine which allows a high-level C program to use the lower-level OS-9 system call F$TLink. The 'tlink' routine, which uses the OS-9 C Compiler's inline assembler, is shown below:

```
/*  Assigns/unassigns trap handler modules to software trap exception vectors.  */

tlink(trap_num,addit_mem,mod_name)
register
int
    trap_num,    /* Trap number to be assigned to the trap module.          */

    addit_mem,   /* Additional memory to be assigned to the trap module
                    over and above the amount already declared for the module.  */

    mod_name,    /* Address of the null-terminated name string for the trap module.
                    If this is 0, or points to a 0, the trap number becomes available
                    for reassignment.                                         */
{
@   MOVE.L   D4,D0       trap number
@   MOVE.L   D5,D1       additional memory request
@   MOVEA.L  D6,A0       module name pointer
@   OS9      F$TLink     install trap module
@   EXT.L    D1          error code or 0 is returned in D1.W
@   MOVE.L   D1,D0       return error code or 0 in D0.L
}
```

90

'Generic' Format Trap Modules

The 'tlink' routine uses three register variables: trap_num , addit_mem , and mod_name. According to page 3-3 of the OS-9 C Compiler User's Manual, the OS-9 C Compiler uses registers D4-D7, and A2-A4 for register variables. For this routine it was determined that the first register variable declared was assigned to register D4, the second to register D5, and the third to register D6. The first action of the 'tlink' routine is to set up the registers for the F$TLink system call (described on page 14-44 of the OS-9/68000 Operating System Technical Manual) by moving the input parameters from their register variable locations to the registers used by the F$TLink call. The error code returned by the F$TLink call is returned in the D0 register.

The version of the 'tlink' routine included on the MACH2 distribution disk contains much more extensive error handling. The version shown has been trimmed down for demonstration purposes.

The 'ttcall' Routine

The 'TRAPn' 68000 assembly language instruction is used to 'call' a trap handler module. This code fragment shows how a trap handler module would be 'called' from assembly language:

```
TRAP #5
DC.W 1
```

These instructions would call the trap handler module currently assigned to software exception vector #5. The trap handler module would be passed a selector value of 1. Since the 'TRAP' instruction is only accessible from assembly language, the 'ttcall' routine was created to allow C programs to call trap handler modules:

```
/*   Allows C programs to call a trap handler module using any software
     exception vector number and any selector value.                    */

ttcall(trap_num,selector)

register
int trap_num,    /* Trap number to be assigned to the trap module.      */

    selector,    /* The selector value which will be passed to the trap module. */

{
@   MOVE.W #$4E75,-(A7)    lay down an 'RTS' instruction
@   MOVE.W D5,-(A7)        lay down the selector value
@   ORI.W  #$4E40,D4       calculate the trap opcode value
@   MOVE.W D4,-(A7)        lay down the 'TRAPn' opcode
@   JSR    (A7)            execute trap call by 'jumping' to the instruction
@   ADDQ.L #6,A7           reclaim stack space
}
```

91

'Generic' Format Trap Modules

The 'ttcall' routine also uses register variables. When the 'ttcall' is executed, the D4 register will contain the user's desired trap number and the D5 register will contain the selector value the user wishes to pass to the trap handler module.

The 'ttcall' routine makes no assumptions about the software exception vector number to be used for the trap handler module call or about the selector value which is to be passed to the trap handler module. Both of these parameters are passed into the 'ttcall' routine. In order to be this flexible, 'ttcall' must construct the 'TRAP' and 'DC.W' assembly language instructions on the system stack each time it executes.

These diagrams should help explain 'ttcall's actions.

ttcall(ADTrap,AD2)

'ttcall' instructions		system stack	constructed instructions
MOVE.W #$4E75,-(A7)	addr+4	4E75	RTS
MOVE.W D5,-(A7)	addr+2	2	DC.W 2
ORI.W #$4E40,D4 MOVE.W D4,-(A7)	addr	4E45	TRAP #5

The left column shows the 'ttcall' assembly instructions, the middle column shows how the assembly instructions affect the system stack, and the right column shows what instructions the values on the system stack represent. After the instructions have been constructed on the stack, they are executed and the stack is cleaned up.

Summary

Using MACH2 generated trap modules from programs written in other languages is a three step process. First, some means of accessing the OS-9 F$TLink system call from within the language must be found. In the example, the OS-9 C Compiler did not provide a high level function which allowed access to the F$TLink call, so the inline assembler was used. Next, some means of executing a 68000 'TRAP' instruction must be found. The OS-9 C Compiler's inline assembler was also used for this purpose in the example program. Finally, the register usage and parameter passing methods of the language must be determined so that the calling program and the MACH2 'generic' trap handler module will interact successfully.

'Generic' Format Trap Modules

Glossary

!   (n a -)

" store "

Format: number address !

Action: Stores the 32-bit number at the specified address.

Example: The following example shows that the ! operator takes the full 32-bit value off the stack and stores it in a 32-bit location in memory :

HEX   <cr> ok <$0>

VARIABLE   STORAGE   <cr> ok <$0>

3CCD4   STORAGE   !   <cr> ok <$0>   The number 3CCD4 as it appears on the stack.

1 byte	1 byte	1 byte	1 byte
0 0	0 3	C C	D 4

Bytes in memory :

`00 03 CC D4`

lower memory --> higher memory

STORAGE

! takes all four bytes (32-bits) from the stack and stores it starting at the specified address.

For Assembly Language Programmers:

```
CODE ! ( n a - )
        MOVE.L     (A5)+,A0
        MOVE.L     (A5)+,(A0)
        RTS
END-CODE
MACH
```

See also: @ , W! , W@ , C! , C@

G-2

Memory
Operator

"

" quote "

Format:	" ccc"
Action:	Converts the string delimited by the quotation marks to both counted string format (length byte followed by the string itself, the string cannot be longer than 255 characters) and "C" string format (string followed by a null byte) and returns the address of the counted string (the address of the length byte). To get the address of the "C" string you must add 1 to the address returned (to skip over the length byte). The string is stored in the dictionary. The leading " must be followed by at least one space.

Example: The diagram shows how the string compiled by " below would look in the dictionary:

```
HEX   <cr> ok <$0>
" Hello" ok <$1>
.  <cr> 82A04 <$0>
```



```
05 68 65 6C 6C 6F 00 00 FE
```

82A04:

                              alignment byte
                              (odd length strings only)

            H   e   l   l   o

length byte   null byte

If a program uses a string several times, the string should be created upon start-up and its address saved in a variable for future references:

    VARIABLE StringPtr   <cr>ok <0>

    : MakeString   " Reusable String"   StringPtr ! ;   <cr>ok <0>

    StringPtr @ COUNT TYPE   <cr> Reusable String ok <0>

To get the address of a "C" string add 1 to the address returned by " :

    " C-string" 1+   <cr> ok <1>

See also: ." , COUNT , TYPE

G-3

**Character
I/O**

# #

Format:	<# . . . . . # . . . . . #>
Action:	Using the number on top of the stack, # converts one digit to ASCII and inserts it into the formatted ASCII string being constructed in the PAD. # will always insert a digit when it is executed. # must be used within <# and #>.

Example: The word $String takes numbers off the stack and prints them out in a dollars and cents format, $XXX.XX :

: $String (n -) <# # # ASCII . HOLD #S ASCII $ HOLD #> TYPE ; <cr> ok <0>

77693 $String <cr> $776.93 ok <0>

The first # is used to put the ones digit in the string. The second # is used to put the tens digit into the string. The ASCII . HOLD inserts the decimal point. The #S inserts any remaining numbers into the string in the dollars section of the formatted string. The ASCII $ HOLD puts a dollar sign in front of the entire string.

PHONE# takes numbers off the stack and prints them out in phone number format :

: PHONE# (n -) <# # # # # ASCII - HOLD # # # #> TYPE ; <cr> ok <0>

1234567   PHONE#   <cr> 123-4567 ok <0>

Each # inserts one phone digit into the formatted string.

See also: <# , #S , HOLD , SIGN , #>

G-4

Number
I/O

#>   (n1 - a n2)

Format: <# .. (any number of formatting operators) .. #>

Action: Drops the number from the top of the stack (the remains of
                 the number which was to be converted into a formatted ASCII
                 string--will be zero if all digits were used up) and sets up
                 the stack for TYPE by leaving the count byte (n2) and the
                 string address (a) on the stack.

Example:

< #	Start a new formatted number string.
#	Insert the next digit of the number being printed into the formatted number string.
# S	Insert all remaining significant digits of the number into the formatted number string.
HOLD	Insert the character on the stack into the formatted number string.
SIGN	Insert a minus sign into the formatted number string if appropriate.
# >	Terminate the formatted number string. The string in now ready for printing (it is set up for TYPE).

See also: <# , # , #S , HOLD , SIGN

G-5

Number
I/O

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

#OUT   (- n)

" number-out "

Format: #OUT

Action: Returns the number of characters that have been typed
                 out on the current line. The result is only valid when the
                 screen is the current output device.

Example: In the following example #OUT is used to show that 13
                 characters have been output on the current line:

: TEST (-)   " Testing." COUNT TYPE #OUT . ;   <cr> ok <0>

TEST<cr> Testing.13 ok <0>

Character
I/O

#S

" sharp-s "

Format:	<# . . . . . #S . . . . . #>
Action:	Keeps converting digits from the number on top of the stack to ASCII and inserting them into the formatted string being constructed until the number on top of the stack is zero. #S will always produce at least one digit, even if the number on top of the stack is initially zero (in which case #S will produce a 0 and terminate). #S must be used within <# and #>.

Example: #S is used in the definition of U. . U. takes a number from the stack and prints it in its entirety as an unsigned number :

: U. (n -) <# #S #> TYPE SPACE ; <cr> ok <0>

1234567   <cr> 1234567 ok <0>

The #S continually takes digits from the number on top of the stack and inserts them in the formatted string being constructed until the number on top of the stack is reduced to zero.

The definition of #S is:

```
:  #S
     BEGIN
         #          ( Convert one digit from the number on top of the stack
                       to an ASCII character and insert it in the formatted
                       string being constructed. )
         DUP
         0 =        ( Has the number on top of the stack been reduced to
                       zero- indicating that there are no digits left to convert- ? )
     UNTIL  ;        ( If so, leave #S .)
```

See also: <# , # , HOLD , SIGN , #>

G-7

Number
I/O

#TIB (- a)

" number-t-i-b "

Format:	#TIB
Action:	Returns the address of the system variable that contains the number of characters currently in the terminal input buffer. WORD uses #TIB as it parses a line in the text input buffer to determine when it has reached the end of the buffer. BLK, >IN, #TIB, and TIB are the four system variables responsible for maintaining control of the input stream.

Example:

QUERY is the word FORTH uses to get its terminal input. Execution of QUERY terminates when either a carriage return is received or when the maximum capacity of the TIB has been reached (the TIB can hold up to 72 characters). When QUERY terminates it will put the number of characters it received into #TIB :

#TIB @ . <cr> 8 ok <0>

```
#TIB  @  .  <cr>  8 ok <0>
▲▲▲▲▲▲▲▲
       │8
      │7
     │6
    │5
   │4
  │3
 │2
1
```

(The 8 indicates that 8 characters were received during the last execution of QUERY--assuming that a carriage return was hit immediately after the . was typed.)

See also: BLK , TIB , >IN , WORD

System/Local
Variable

$

Format:	$ <command line>
Action:	Used to interactively execute OS-9 utility commands. The command line should follow $ and be terminated by a carriage return.

Example:
To interactively check the amount of available, unused memory you can use the OS-9 utility command MFREE:

$ mfree   <cr>

Current total free RAM: 203.75 K-bytes
ok <0>

OS-9 Note: To execute OS-9 utility commands MACH2 creates a child process (using F$Fork) and has the child process execute the command. MACH2 then deactivates itself (using F$Wait) until the child process terminates execution.

See also: TCALL

OS-9
Interface

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

' _____ **(- a)** _____ " tick "

Format:	' <name>
Action:	Returns the address of the executable code for the <name> which follows it in the current input stream. ' searches for <name> using the current dictionary search order. If <name> is found, ' leaves the parameter field address for the word on the stack. If <name> is not found, an error message is issued.
	' should only be used interactively. Use ['] inside of colon definitions.
Example:	' may be used with EXECUTE. EXECUTE will execute the code starting at the address passed to it:
	3 ' DUP EXECUTE . . <cr> 3 3 ok <0>

See also: ['] , FIND , EXECUTE

Compilation Word

'= (- a1 a2 f | a f)

" tick-equal "

Format:	'= <name>
Action:	Tries to find <name> using the current dictionary search. If <name> is found, '= returns first <name>'s parameter field address (a1), link field address (a2) , and a false (0) flag. If <name> is not found, '= returns the address of a string containing the name and a true (non-zero) flag.

Example: DUP will be found in the current search order :

ONLY   FORTH <cr> ok <0>
HEX   <cr> ok <$0>

'= **DUP** <cr> ok <$3>

.S <cr> 79F3A 17A9E 0 ok <$3>

JUNK will not be found :

'= **JUNK** <cr> ok <$2>

.S <cr> 180A0 1 <- TOP ok <$2>

See also: ' , ['] , FIND

G-11

Compilation Word

(

" right-paren "

Format: (...comments...)

Action: Begins a comment. All words between the left
                 paren and the corresponding right paren will be
                 ignored. '(' must be followed by at least one space.
                 Nested parentheses are accepted.

Example: The following definition contains several uses of "(" for comments:

```
: LF-Filter ( address length --   )
    length 0 DO                 ( step through entire string )
        address I + C@          ( get a character )
        7F AND                  ( throw away the 8th bit )
        address I + C!          ( store the altered character )

        address I + C@ A =      ( check for linefeeds )
        IF
            SP address I + C!   ( replace linefeeds with carriage returns )
        THEN
    LOOP
    ( address length TYPE       ( eventually we will type the string out ... ) )
    ;
```

Notice that the last line contains nested parentheses.

The word \ may be used for commenting out a single line. (may be used to
comment out any number of lines.

See also: .(, \

FORTH
Tool

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

*   (n1 n2 - n3)

" times "

Format: n1 n2 *

Action: Multiplies n1*n2, leaves 32-bit result on top of the stack.

Example: 4 5 * <cr> ok <1>

. <cr>20 ok <0>

See also: UM*, 2*, */, */MOD

G-13

**Arithmetic
Operator**

*/ (n1 n2 n3 - n4)

" star-slash "

Format: n1 n2 n3 */

Action: Multiplies n1*n2 to get a 64-bit result. Then divides by n3 to produce a 32-bit result.

Example: 1000000 4 8 */ <cr>ok <0>

. <cr> 500000 ok <0>

8 →

4 →

1000000 →

500000 →

See also: *, /, */MOD

Arithmetic Operator

G-14

*/MOD   (n1 n2 n3 - n4 n5)

" star-slash-mod "

Format:	n1 n2 n3 */MOD
Action:	Multiplies, then divides - (n1*n2)/n3 - using a 64-bit intermediate result. Leaves the 32-bit quotient on top of the stack and the 32-bit remainder immediately below the quotient.

Example: 10000   5   20000   */MOD <cr> ok <2>

.S   <cr> 10000 2   <- TOP ok <2>

20000 →

5 →

10000 →

2 →

10000 →

See also: /MOD, */, MOD

G-15

Arithmetic
Operator

+   (n1 n2 - n3)

" plus "

Format: n1 n2 +

Action: Adds.   Replaces the two numbers on top of the stack
                           with their sum.

Example: 3 5 + <cr> ok <1>

                           . <cr>

For Assembly Language Programmers:

```
CODE + ( n1  n2  -  n3 )
      MOVE.L    (A5)+,D0
      ADD.L     D0,(A5)
      RTS
END-CODE
MACH
```

See also: 1+ , 2+ , D+

Arithmetic
Operator

+!   (n a -)

" plus-store "

Format:	number address +!
Action:	Adds the 32-bit value to the contents of the specified address.

Example:

The following example shows that the +! operator takes the 32-bit value from the stack and adds it to the 32-bit value stored at the specifed address :

HEX   <cr> ok <$0>

VARIABLE   STORAGE   <cr> ok <$0>

1 STORAGE ! <cr> ok <$0>   (Put a 1 into STORAGE)

7 STORAGE +! <cr> ok <$0>   This is how the 7 looks on the stack.

1 byte	1 byte	1 byte	1 byte
00	00	00	07

Bytes in memory :

									00	00	00	08						

lower memory -> higher memory

STORAGE

+! took the 7 on top of the stack and added it to the 1 stored in the STORAGE variable. Every number involved was treated as a 32-bit value.

For Assembly Language Programmers:

```
CODE +! ( n a - )
        MOVE.L    (A5)+,A0
        MOVE.L    (A5)+,D1
        ADD.L     D1,(A0)
        RTS
   END-CODE    MACH
```

See also: ! , @ , W! , W@ , C! , C@

G-17

Memory Operator

+>   (n1 -)

" plus-to "

Format: n1 +> <name of local variable>

Action: Adds the number on top of the stack to the local variable
                     specified by name.

Example: We will use '+>' in this example to help sum the numbers
                     from 0 - 9:

```
: TENSUM {   | sum -- sum }
    0 -> sum
    10  0  DO
       I +> sum
    LOOP
    sum  . ; ok <0>

TENSUM   <cr> 45 ok <0>
```

See also: -> , ^ , {

G-18

System/Loc
al
Variable

+LOOP (n -)

Format: limit index
DO
(code to be executed each time through loop)
increment value
+LOOP

Action: +LOOP is used when incrementing the loop index by a
number other than one is desired.
+LOOP expects to find a limit and index on the return stack
and the desired incrementing value on the parameter
stack. Each time +LOOP is executed it adds the desired
increment to the loop index value and compares the
new index value to the loop limit. If a positive increment
value is specified, the loop will continue to be executed
until the index value is greater than or equal to the limit
value. If a negative increment value is specified, the
loop will continue to be executed until the index is less
than or equal to the limit. When the condition for loop
termination has been reached, +LOOP will remove the limit
and index from the return stack and allow program execution
to continue on to the code which follows the +LOOP.

Example: STEP_UP uses a positive loop index with +LOOP :

```
: STEP-UP
    30  0  DO
                    I .
            5
            +LOOP  ;  <cr> ok <0>

STEP-UP   <cr> 0 5 10 15 20 25 ok <0>
```

STEP-DOWN uses a negative loop index with +LOOP :

```
: STEP-DOWN
        -10  0  DO
                    I .
            -2    .
            +LOOP  ;  <cr>  ok <0>

STEP-DOWN   <cr> 0 -2 -4 -6 -8 -10 ok <0>
```

See Also: DO , LOOP , I , J

Program
Control
Structure

G-19

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z{|}~

,_____(n -)_____
                                                                " *comma* "

Format:	32-bit-value ,
Action:	Lays the 32-bit number in the dictionary starting at the next available memory location in the dictionary. , will first check to make sure it is on a word boundary and adjust the HERE pointer if necessary. The pointer to the next available dictionary location, the HERE pointer, is incremented by 4 bytes.

<u>**Example:**</u>   In the following example, CREATE is used to make a dictionary entry for TABLE. Then, , is used to store four 32-bit values starting at the parameter field address of TABLE :

CREATE   TABLE   <cr> ok <0>

10 , 100 , 1000 , 10000 , <cr> ok <0>

TABLE   @   .   <cr> 10 ok <0>

TABLE   12   +   @   .   <cr> 10000 ok <0>

<u>For Advanced Programmers:</u>
An important fact to notice in the above example is that CREATE creates the dictionary entry in the area where the names are stored (in the names space) while ',' stores values in the area where the executable code for the dictionary entries is kept (in the code space).

See also: W, , C, , ALLOT , HERE

G-20

Compilation
Word

— (n1 n2 - n3)

" minus "

Format: n1 n2 -

Action: Subtracts n1-n2 and leaves the result on top of the stack.

Example: 100 35 - <cr> ok <1>

                  . <cr> 65 ok <0>

For Assembly Language Programmers:


```
CODE -  ( n1 n2 - n3 )
    MOVE.L    (A5)+,D0
    SUB.L    D0,(A5)
    RTS
END-CODE
MACH
```


See also: 1- , 2-

G-21

Arithmetic
Operator

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z{|}~

-> (n -)

" save-to "

Format:	number -> \<name>
Action:	Stores the number on top of the stack into the specified local variable name or named input parameter.

Example:

Local variables could be used to solve this equation for any specified X, Y combination: (X*Y)+(X-Y)=N.

```
: EQUATION1 { X  Y  |  X1  Y1  -- N }
    X Y * -> X1
    X Y - -> Y1
    X1  Y1  +  ;  <cr> ok <0>

10  5  EQUATION1  .  <cr> 55 ok <0>
```

The contents of a local variable or named input parameter are put on the stack by typing their name. Values are stored in a local variable or named input parameter by using the -> operator.

X and Y are named input parameters. They require a value from the stack on input.

X1 and Y1 are local variables. They are used for temporary storage.

N is a comment, indicating stack notation.

See also: +> , ^ , { , RECURSIVE

G-22

System/Local Variable

-TRAILING (a n1 - a n2)

" minus-trailing "

Format:	string-address original-length -TRAILING
Action:	Reduces the character count of a string by eliminating any trailing spaces. Expects on the stack the length in bytes and address of the string. Leaves the address of the string and the new reduced length of the string.

Example:

VARIABLE STRING 36 VALLOT <cr> ok <0>

STRING 40 EXPECT <cr> string with trailing spaces   ↑   <cr> ok <0>

(10 spaces)

STRING 40 TYPE <cr> string with trailing spaces   ↑   ok <0>

(spaces still included in string)

STRING 40 -TRAILING TYPE <cr> string with trailing spaces ok <0>
↑

(string count has been reduced to eliminate trailing spaces)

See also: TYPE , COUNT

G-23

Character
I/O

.   (n -)

                                                               " dot "

Format: n .

Action: Prints the signed number on top of the stack on the screen followed by one space. . checks the sign bit (bit 31) of the number on the stack to determine if the minus sign should be printed.

Example:

    3 **.**   **<cr>** 3 ok <0>

    -3 **.**   **<cr>** -3 ok <0>

U. , on the other hand, treats all numbers as unsigned numbers :

    HEX   **<cr>** ok <$0>

    -3 **U.**   **<cr>** FFFFFFFD ok <$0>

 The definition of . is :

 : . (n -) DUP ABS <# #S SIGN #> TYPE SPACE   ; **<cr>** ok <0>

See also: **U. , SIGN**

G-24

Number I/O

```
 ! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~
```

" "

.

———

`" dot-quote "`

Format: ." ccc"

Action: Used inside of a colon definition to compile a string which
                 will be typed out at execution time. The ." must have a space
                 after it. The final " is not included in character string. A ."
                 string may be a maximum of 255 characters in length.

                 .(should be used if a string is to be typed out interactively.

Example: : HELLO ." Hello !" ; <cr> ok

                 HELLO   <cr> Hello ! ok

See also: .(, "

Chararacter
I/O

.(

" dot-paren "

Format:	.(...ccc...)
Action:	Immediate word which types the character string delimited by the parentheses out to the current output device. When used inside of a colon definition the string will be printed out during compile time. When executed immediately the string will be typed out immediately. The .(must be followed by a space.

Example: .(is often used to print out messages to indicate what part of a program is currently being compiled. For example, the following is a listing of the program file 'INIT-TABLES.

```
\  This is the file named 'INIT-TABLES'.

CR
.(  Initializing sine table. )
:  INIT-SINE-TABLE  ;

CR
.(  Initializing cosine table. )
:  INIT-COSINE-TABLE  ;

CR
.( Initializing log table. )
:  INIT-LOG-TABLE    ;
```

·The .(messages imbedded in the 'INIT-TABLES' file are printed out when 'INIT-TABLES' is loaded:

```
INCLUDE" INIT-TABLES"   <cr>
Initializing sine table.
Initializing cosine table.
Initializing log table.  ok <0>
```

See also: (

Character
I/O

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

.R (n1 n2 -)

" dot-r "

Format:	*number-to-be-printed field-width .R*
Action:	Prints the signed number, right-justified in a field with the specified width. If the number is wider than the field, no leading blanks will be printed.

Example:

```
:   TWO-FIELDS   CR
      2345  10 .R  CR
      23      10 .R  CR    ; <cr> ok <0>
```

TWO-FIELDS <cr>

```
          2345
       ↑
6 spaces
```

```
         23  ok <0>
      ↑
8 spaces
```

See also: . , U.

G-27

Number
I/O

.S

Format: . S

Action: Displays the items on the parameter stack without
                 destroying the contents of the stack. The number on
                 top of the stack will be the rightmost number displayed
                 on the screen.

Example: 10 20 30 40 50 60 70 <cr> ok <7>

                 .S <cr> 10 20 30 40 50 60 70 <- TOP ok <7>

See also: DEPTH.

G-28

FORTH
Tool

/ (n1 n2 - n3)

" divide "

Format: n1 n2 /

Action: Divides n1/n2. Quotient is an 32-bit result rounded
               towards zero.

Example: In the first example the division works out evenly -

               50 5 / <cr> ok <1>

               . <cr> 10 ok <0>

               In this example, the division does not work out evenly -

               25 8 / <cr> ok <1>

               . <cr> 3 ok <0>

See also: 2/ , /MOD , */ , */MOD , MOD

G-29

Arithmetic
Operator

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

/MOD (n1 n2 - n3 n4)

" slash-mod "

Format: n1 n2 /MOD

Action: Divides n1/n2, leaving the 32-bit quotient on top of the stack
and 32-bit remainder immediately below the quotient.

Example: 17 4 /MOD <cr> ok <2>

.S <cr> 1 4 <- TOP ok <0>

See also: MOD , /

G-30

Arithmetic
Operator

0< (n - f)

" zero-less-than "

Format: n1 0<

Action: Returns a true flag if n is less than 0 (if n is negative).

Example: 4 0< <cr> ok <1>

. <cr> 0 ok <0>

-2 0< <cr> ok <1>

. <cr> -1 ok <0>

For Assembly Language Programmers:

```
        CODE  0< ( n - f )

            MOVEQ.L   #0,D0
            TST.L     (A5)+
            BGE.S     @1
            MOVEQ.L   #-1,D0
@1          MOVE.L    D0,-(A5)
            RTS
        END-CODE
        MACH
```

See also: 0> , 0=

G-31

Comparison
Operator

0=   (n - f)

" zero-equal "

Format:	n 0=
Action:	Returns a true flag if the number on top of the stack is a zero or a false flag if the number is non-zero.

Example:

5 0= <cr> ok <1>   0 0= <cr> ok <1>

. <cr> 0 ok <0>   . <cr> ok -1 <0>

For Assembly Language Programmers:

```
CODE 0=   ( n  -  f )

     MOVEQ.L  #0,D0
     TST.L    (A5)+
     BNE.S    @1
     MOVEQ.L  #-1,D0
@1   MOVE.L   D0,-(A5)
     RTS
END-CODE
MACH
```

See also: 0<, 0>

G-32

Comparison
Operator

1+   (n1 - n2)

Format: n 1+

Action: Adds one to the number on top of the stack.

Example: 3 1+ <cr> ok <1>

. <cr> 4 ok <0>

For Assembly Language Programmers:

```
CODE 1+  ( n1 - n2 )
     ADDQ.L   #1,(A5)
     RTS
END-CODE
MACH
```

See also: 1- , + , 2+

Arithmetic Operator

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

1-   (n1 - n2)

" one-minus "

Format: n 1-

Action: Subtracts one from the number on top of the stack.

Example: 10 1- <cr> ok <1>

              . <cr> 9 ok <0>

For Assembly Language Programmers:

```
CODE 1-  ( n1 - n2 )
      SUBQ.L    #1,(A5)
      RTS
END-CODE
MACH
```

See also: 1+ , - , 2-

Arithmetic
Operator

2* (n1 - n2)

" two-times "

Format: number 2*

Action: Multiplies the number on top of the stack by two.

Example: 6 2* <cr> ok <1>

. <cr> 12 ok <0>

6 DUP BINARY . <cr> 110 ok <%1>

2* . <cr> 1100 ok <%0>

For Assembly Language Programmers:

```
CODE 2* ( n1 - n2 )

    MOVE.L  (A5),D0
    ADD.L   D0,(A5)
    RTS

END-CODE
MACH
```

See also: 2/, *, /, 2+, 2-, BINARY, DECIMAL

Arithmetic
Operator

2+   (n1 - n2)

" two-plus "

Format: n 2+

Action: Adds two to the number on top of the stack.

Example:

7 2+ <cr> ok <1>

. <cr> 9 ok <1>

For Assembly Language Programmers:

```
CODE  2+   ( n1 - n2 )
      ADDQ.L    #2,(A5)
      RTS
END-CODE
MACH
```

See also: 2- , 1+

Arithmetic
Operator

2-   (n1 - n2)

" two-minus "

Format:	n 2-
Action:	Subtracts two from the number on top of the stack.

Example:

5 2- <cr> ok <1>

. <cr> 3 ok <0>

For Assembly Language Programmers:

```
CODE  2-   ( n1 - n2 )
      SUBQ.L   #2,(A5)
      RTS
END-CODE
MACH
```

See also: 2+ , 2* , 1-

G-38

Arithmetic Operator

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

2/   (n1 - n2)
_____
" two-divide "

Format: n 2/

Action: Divides the number on top of the stack by two by
                arithmetically shifting the number right one bit.
                Returns integer result rounded towards zero.

Example: 4 2/ <cr> ok <1>
                . <cr> 2 ok <0>

                5 2/ <cr> ok <1>
                . <cr> 2 ok <0>

For Assembly Language Programmers:

        CODE 2/ (n1 - n2)
              MOVE.L   (A5),D0
              ASR.L   #1,D0
              MOVE.L   D0,(A5)
              RTS
        END-CODE
        MACH

See also: 2+, 2-, /

G-39

Arithmetic
Operator

2DROP   (n1 n2 -)

" two-drop "

Format: n1 n2 2DROP

Action: Removes two numbers from the top of the parameter stack.

Example: **30   40   2DROP**   <cr> ok <0>

For Assembly Language Programmers:

```
    CODE  2DROP  ( n1  n2  -  )
          ADDQ.L   #8,A5
           RTS
     END-CODE
     MACH
```

See also: DROP , 2SWAP , 2OVER , 2DUP

**Stack
Manipulation**

2DUP   (n1 n2 - n1 n2 n1 n2)

" two-dupe "

Format: n1 n2 2DUP

Action: Duplicates the two single-length (32-bit) numbers
                  on top of the parameter stack.

Example: 30   40   2DUP   <cr> ok <4>

For Assembly Language Programmers:

```
        CODE  2DUP   (  n1  n2  -  n1  n2  n1  n2  )
            MOVE.L    4(A5),-(A5)
            MOVE.L    4(A5),-(A5)
            RTS
        END-CODE
        MACH
```

See also: DUP , 2SWAP , 2OVER , 2DROP

G-41

Stack
Manipulation

2OVER   (n1 n2 n3 n4 - n1 n2 n3 n4 n1 n2)

Format: n1 n2 n3 n4 2OVER

Action: Puts a copy of the second pair of numbers on the stack
               on top of the stack.

Example: 20 40 30 50 2OVER <cr> ok <6>

               .S <cr> 20 40 30 50 20 40 ok <6>

See also: OVER , 2SWAP , 2DUP , 2DROP

Stack
Manipulation

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z{|}~

2SWAP   (n1 n2 n3 n4 - n3 n4 n1 n2)

" two-swap "

Format: n1 n2 n3 n4 2SWAP

Action: Switches the top two pairs of numbers on the parameter stack.

Example: 30 40 50 60 2SWAP <cr> ok <4>

A high-level definition of 2SWAP is :

    : 2SWAP (n1 n2 n3 n4 - n3 n4 n1 n2)
            3 ROLL 3 ROLL ; <cr> ok <0>

See also: SWAP, 2DUP, 2DROP, 2OVER

G-43

Stack
Manipulation

:
_____

" colon "

Format:	: <name> (contents of colon definition) ;
Action:	Defining word used to create new dictionary entries. The compile-time action of : involves creating a new dictionary entry using <name>, putting the system into the compilation state, and setting the smudge bit of the new definition so the definition will not be visible until it is completed. RECURSIVE is used when a definition needs to reference itself.

Example: A colon definition may be as simple as this definition (which does nothing):

: **NOTHING** ; ok <0>

Or it may contain a series of other FORTH words:

: **GREETINGS**
   BEGIN
    ." Hello !"
   CR
   ?TERMINAL
   UNTIL
; ok <0>

To determine to which vocabulary a new definition will be appended, use ORDER.
In the following example, the results of ORDER indicate that currently, the ASSEMBLER
vocabulary is searched first, and then the FORTH vocabulary. Any new definitions
will be appended to the FORTH vocabulary :

ONLY FORTH   DEFINITIONS <cr> ok <0>
ALSO   ASSEMBLER <cr> ok <0>
ORDER <cr>

Search Order : ASSEMBLER FORTH
Definitiions   : FORTH
 ok <0>

See also: ; , EXIT , STATE , RECURSIVE

G-44

Defining
Word

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

;
_____

" semi-colon "

Format: : <name> (contents of colon definition) ;

Action: Immediate word which compiles the run-time code for exit--an
                   RTS, at the end of the new colon definition. ; then clears
                   the smudge bit so that the word may be found in dictionary
                   searches and puts the system back into the execution state.

See also: : , EXIT

G-45

Compilation
Word

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

;CODE

" semi-colon-code "

Format:

: name
    CREATE   ... compiling behavior...
      ;CODE   ...assembly language run-time behavior...
    END-CODE

Action:

Used in the definition of a new defining word. ;CODE is similar to DOES> except that it allows the run-time code to be expressed in assembly language. During compilation of the defining word ;CODE adds the ASSEMBLER to the search order. During the run-time of the daughter word ;CODE will leave the parameter field address on the subroutine stack. When ;CODE is used to define run-time behavior, the defining word must end with an END-CODE.

See also: DOES> , END-CODE

G-46

Compilation Word

<   (n1 n2 - f)

" less-than "

Format: n1 n2 <

Action: Compares the two numbers on top of the stack. Returns a true flag if n1 is less than n2, and a false flag if n1 is not less than n2.

Example:

4 8 < <cr> ok <1>

. <cr> -1 ok <0>

4 2 < <cr> ok <1>

. <cr> 0 ok <0>

4 4 < <cr> ok <1>

. <cr> 0 ok <0>

For Assembly Language Programmers:

```
CODE <    ( n1  n2  -  f )
       MOVEQ.L    #0,D0
       CMPM.L     (A5)+,(A5)+
       BGE.S      @1
       MOVEQ.L    #-1,D0
@1     MOVE.L     D0,-(A5)
       RTS
END-CODE
MACH
```
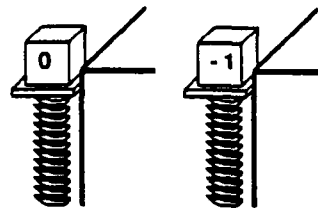
See also: >, =

G-47

Comparison Operator

<#

_____
" bracket-sharp "

Format: n <# (any number of formatting operators) #>

Action: Signifies the beginning of the binary-number to formatted-
ASCII-string conversion process. <# sets up a pointer
to point to the memory location where the first character
(the right-most character) in the formatted ASCII string will
be placed. The number to be converted should be on top of the
stack since any succeeding formatting operators (# , #S ,
HOLD , SIGN , #>) perform their operations on the number
on top of the stack.

Example: DATE# prints the number on top of the stack
out in a date format, month/day/year :

: DATE# (n -) <# # # ASCII / HOLD # # ASCII / HOLD # # #> TYPE ; <cr> ok <0>

100961 DATE# <cr> 10/09/61 ok <0>

The PAD is the area used to construct the formatted ASCII string. The first character,
the right-most digit, is placed in the last byte position in the PAD. A formatted string
may contain up to 84 characters, which is the capacity of the PAD.

0 bytes   84 bytes

Beginning of the PAD.   This is where the first ASCII character
in the formatted string will be placed.

See also: # , #S , HOLD , SIGN , #>

G-48

Number
I/O

<>   (n1 n2 - f)

" not-equal "

Format: n1 n2 <>

Action: Compares the two numbers on top of the stack. Returns
                    a true flag if n1 is not equal to n2, and a false flag if n1 is
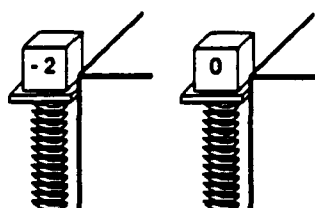   equal to n2.

Example:

4 8 <> <cr> ok <1>   4 4 <> <cr> ok <1>

. <cr> -1 ok <0>   . <cr> 0 ok <0>

For Assembly Language Programmers:

```
        CODE <>   ( n1  n2  -  f )
                MOVEQ.L    #0,D0
                CMPM.L     (A5)+,(A5)+
                BEQ.S      @1
                MOVEQ.L    #-1,D0
        @1      MOVE.L     D0,-(A5)
                RTS
        END-CODE
        MACH
```
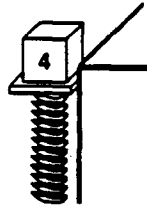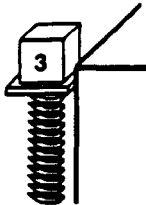
See also: > , = , <

G-49

Comparison
Operator

=   (n1 n2 - f)

" equal "

Format: n1 n2 =

Action: Compares the two numbers on top of the stack. Returns
                            a true flag if they are equal or a false flag if they are not
                            equal.

Example:

5 5 =   <cr> ok <1>   5 6 =   <cr> ok <1>

. <cr> -1 ok <0>   .<cr> 0 ok <0>

For Assembly Language Programmers:

```
        CODE =    ( n1  n2  -  f )
                MOVEQ.L     #0,D0
                CMPM.L      (A5)+,(A5)+
                BNE.S       @1
                MOVEQ.L     #-1,D0
        @1      MOVE.L      D0,-(A5)
                RTS
        END-CODE
        MACH
```

See also: < , >

G-50

Comparison
Operator

>   (n1 n2 - f)

" greater-than "

Format: n1 n2 >

Action: Compares the two numbers on top of the stack. Returns
                           a true flag if n1 is greater than n2. Returns a false flag
                           if n1 is not greater than n2.

Example:

4 8 > <cr> ok <0>

. <cr> 0 ok <0>

8 4 > <cr> ok <0>

. <cr> -1 ok <0>

8 8 > <cr> ok <0>

. <cr> 0 ok <0>

For Assembly Language Programmers:

```
CODE  >  ( n1  n2  -  f )
        MOVEQ.L    #0,D0
        CMPM.L     (A5)+,(A5)+
        BLE.S      @1
        MOVEQ.L    #-1,D0
@1      MOVE.L     D0,-(A5)
        RTS
END-CODE
MACH
```

See also: < , =

G-51

Comparison
Operator

>BODY (a1 - a1)

" to-body "

Format: parameter-field-address >BODY

Action: In a pointer-threaded implementation of FORTH, >BODY takes the code field address (CFA) of a definition and returns its parameter field address (PFA, the address where the executable code for a word actually begins). In MACH 2, which is a subroutine-threaded implementation of FORTH, there are no CFAs. The executable code begins right at the start of the dictionary entry.

The function of >BODY in MACH 2 therefore is that of a high-level no-op. >BODY takes a PFA and returns the PFA.

See also: LINK>BODY , BODY>LINK

G-52

Compilation Word

>IN (- a)

" to-in "

Format: >IN

Action: Puts the address of the system variable whose contents indicate how far WORD has progressed into the current input stream on the stack.

BLK, >IN, TIB, and #TIB are the 4 system variables responsible for maintaining control of the input stream.

Words such as WORD and QUERY alter the value of >IN.

Example:

>IN @   DUP   DROP .   <cr> 6 ok <0>

6

5

(At the time the content of the >IN user variable
was checked, WORD had progressed 6
characters into the current input stream.)

4

3

2

1

See also: WORD , BLK , TIB , #TIB

G-53

System/Local
Variable

>R   (n -)

" to-r "

Format:	n >R
Action:	Moves the number on top of the parameter stack to the top of the return (loop) stack.
NOTE:	MACH 2 has a separate return (loop) stack. I , J , I' may be executed outside of a DO loop to obtain data or indices.

Example: 5 >R <cr> ok <0>

Paramater stack and return
stack before >R -

Parameter stack and return
stack after >R -

Parameter	Return	Parameter	Return
Stack	Stack	Stack	Stack

For Assembly Language Programmers:

```
CODE >R ( n - )

      MOVE.L   D5,(A3)+
      MOVE.L   D6,D5
      MOVE.L   (A5)+,D6
      RTS

END-CODE
MACH
```

See also: R> , R@ , I , J

Stack
Manipulation

?DUP   (n - n (n))

" question-dupe "

Format: n ?DUP

Action: Duplicates the number on top of the stack if it is
                 non-zero.

Example: 4 ?DUP <cr> ok <2>

                 .S <cr> 4 4 <- TOP ok <2>

0 ?DUP <cr> ok <1>
.S <cr> 0 <- TOP ok <1>

For Assembly Language Programmers:

```
CODE ?DUP    ( n - n (n) )
     TST.L    (A5)
     BEQ.S    @1
     MOVE.L   (A5),-(A5)
@1   RTS
END-CODE
MACH
```

See also: DUP , OVER , PICK , DROP

G-55

Stack
Manipulation

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

?FREE

" question-free "

Format:	?FREE
Action:	Outputs information about the remaining available code space, variable space, and names space.

Example:

The following example shows how to use ?FREE. Note that the sizes of each of the spaces are system dependent. The numbers ?FREE produces on your system will probably be much different than those shown below:

```
?FREE   <cr>
Code : 32768
Variable : 12000
Names : 8000
ok <0>
```

G-56

FORTH
Tool

```
! " #$%&'()*+,-./0-9:;<=>?@A-Z[\]^_`a-z{|}~
```

?INCLUDE" (f -)

Format: flag ?INCLUDE" <filename>"

Action: Conditional version of the word INCLUDE". The file
specified by name will only be included if a false flag
(zero) is passed to ?INCLUDE".

?INCLUDE" may not be used within a colon definition.

Example: ?INCLUDE" is normally used during a loading process
to ensure that no files are loaded twice. To check to
see if a file has been loaded you would use FIND to see
if a word in that file has already been loaded into the
dictionary. If the word is found FIND will return a true
(non-zero) flag and an address. When the non-zero flag
is passed to ?INCLUDE" the file will not be included
(loaded). If the word in the file is not found FIND will
return a false (zero) flag and an address. If a false flag is
passed to ?INCLUDE" the specified file will be loaded.
Here is an example of the use of ?INCLUDE":

" File1Stub" FIND   <cr> ok <2>   (Check for word in file)

.S <cr> 76880 0 <-- TOP ok <2>   (0 means the word wasn't found)

SWAP DROP <cr> ok <1>   (Drop the address)
?INCLUDE" File1" <cr>   (The zero on top of the stack
                                     will cause ?INCLUDE to
                                     load File1)

See also: INCLUDE" , FIND

Storage
I/O

G-57

?TERMINAL (- f)

Format:	?TERMINAL
Action:	Checks to see if a key has been pressed. Returns a true (non-zero) flag if a key has been pressed and a false (zero) flag if no key has been pressed.

Example: ?TERMINAL is commonly used as the exit test for a BEGIN...UNTIL loop. Execution of the loop will terminate when a key is pressed :

```
: HELLO
    BEGIN
      ." Hello"  CR
    ?TERMINAL
    UNTIL  ;  <cr> ok <0>
```

HELLO <cr> Hello

Hello
Hello
Hello <cr>
ok <0>

See also: BEGIN , UNTIL

G-58

Character
I/O

@ ___ (a - n) _____

" fetch "

Format:	address @
Action:	Replaces the address on top of the stack with the the long-word (32-bit) value stored at the address.

Example:

The following example shows that the @ operator will return the 32-bit value which is stored starting as the specified address :

```
HEX   <cr> ok  <$0>
VARIABLE   STORAGE   <cr> ok  <$0>
800F0   STORAGE   !   <cr> ok  <$0>
```

↓STORAGE

Bytes in memory :

`[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][00][08][00][F0][ ][ ][ ][ ][ ][ ]`

lower memory -> higher memory

0 0	0 8	0 0	F 0
1 byte	1 byte	1 byte	1 byte

```
STORAGE  @  .  <cr> 800F0 ok <$0>
```

For Assembly Language Programmers:

```
CODE  @  ( a - n )
       MOVE.L     (A5),A0
       MOVE.L     (A0),(A5)
       RTS
END-CODE
MACH
```

See also: ! , W@ , W! , C@ , C!

G-59

Memory
Operator

ABORT

Format: ABORT

Action: Clears the parameter, return, subroutine and floating point
                     stacks, sets the state = 0 (the interpreting state), and returns
                     control to the terminal. No message is issued.

Example: The following is a simple example to demonstrate ABORT's
                     effect :

: TEST IF ABORT THEN ; <cr> ok <0>

23 >R <cr> ok <0>   (Put something on the return stack.)

1 2 3 <cr>ok <3>   (Put things on the parameter stack.)

1 TEST <cr>   <cr> ok <0>   (This caused TEST to ABORT. Had to
                                   hit 2 carriage returns to get the 'ok'
                                   prompt back.)

R@ . <cr> -1 ok <0>   (The return stack has been emptied.
                                   There is no underflow message for
                                   the return stack.)

.S <cr> Empty ok <0>   (The parameter stack has been emptied.)

For Advanced Programmers:

ABORT is a vectored routine. The address of the ABORT routine currently being used
is stored in the system variable ABORT_VECTOR. An example of a custom abort
handling routine is shown on the ABORT_VECTOR glossary page.

See also: ABORT", ABORT_VECTOR , QUIT

Program
Control
Structure

ABORT" (f -)

" abort-quote "

Format:	flag ABORT" abort message"
Action:	If the flag is true, the message contained within the quotes is displayed and then the ABORT command is executed. If the flag is false, no action is taken.

Example:

```
VARIABLE  ErrorFlag   <cr>ok <0>
-1  ErrorFlag  !  <cr> ok <0>

:  ERRORCHECK
    ErrorFlag  @  ABORT" Error !"      ;  <cr>ok <0>

ERRORCHECK   <cr> Error !
```

See also: ABORT, ABORT_VECTOR , QUIT, IF

Program
Control
Structure

G-61

ABORT_VECTOR (- a)

" abort-vector "

Format:	ABORT_VECTOR
Action:	Returns the address of the system variable which contains the address of the routine currently being used to handle system aborts.

Example: Users may install their own custom abort handling routine by storing the address of the routine in the ABORT_VECTOR system variable. The CODE definition 'AbortHandler' is an example of a minimal abort handling routine which performs the standard 'stack-resetting' functions normally performed by ABORT. The initial positions of the subroutine stack pointer, the parameter stack pointer, and the loop/floating point stack pointers are stored in memory following the ABORT_ VECTOR location:

Address of abort handling routine	ABORT_VECTOR
Initial subroutine stack pointer	ABORT_VECTOR+4
Initial parameter stack pointer	ABORT_VECTOR+8
Initial loop stack/fp stack pointer	ABORT_VECTOR+12

```
CODE  AbortHandler  ( - )
      LEA     ABORT_VECTOR+4,A0
      MOVE.L (A0)+,A7          \ get sub stack pointer
      MOVE.L (A0)+,A5          \ get param stack pointer
      MOVE.L (A0)+,A3          \ get loop stack pointer
      MOVE.L A3,D7             \ loop stack and fp stack start
                              \ from the same location
      SUBI.L #16,D7           \ reset fp stack pointer
      PEA     QUIT             \ execute QUIT
      RTS
END-CODE

  :  InstallAbort  ( - )        \ install custom abort routine
      ['] AbortHandler  ABORT_VECTOR !  ;
```

See also: ABORT, ABORT"

G-62

System/Local Variable

ABS   (n - |n|)

" absolute "

Format: n ABS

Action: Leaves the absolute value of the number on top of the stack.

Example: -400 ABS <cr> ok <1>

. <cr> 400 ok <0>

For Assembly Language Programmers:

```
CODE  ABS   ( n - |n| )
        TST.B    (A5)
        BPL.S    @1
        NEG.L    (A5)
@1     RTS
END-CODE
MACH
```

See also: NEGATE

Arithmetic
Operator

AGAIN

Format:	BEGIN (code to be executed endlessly) AGAIN
Action:	AGAIN is the second half of the BEGIN...AGAIN endless loop structure. Whenever AGAIN is reached, it reroutes program execution back to the code which immediately follows the BEGIN. Unless the loop uses an unnatural exit (by using either an EXIT or ABORT command), the BEGIN...AGAIN loop will never terminate.
__Example:__	QUIT, the word which "runs" FORTH, is an example of an endless loop :

```
: QUIT

   BEGIN

      ( clear the return stack )

      ( get input )

      ( examine and act upon the input )

      ." ok"  CR

   AGAIN   ;
```

See also: BEGIN , EXIT , ABORT , UNTIL

Program
Control
Structure

ALLOT   (n -)

Format:	number-of-bytes-to-allot-in-dictionary ALLOT
Action:	Allocates n bytes in the dictionary, starting at the next available dictionary location. The pointer to the next available dictionary location (HERE) is incremented accordingly. Values stored in the dictionary space should be constant values such as arrays of fixed data. Space for values which will change during program execution should be allocated using VALLOT.
Example:	ALLOT is commonly used for creating arrays of values which require time-consuming computations such as tables of angle functions or square roots. Rather than calculate such values as needed in a program, it may be feasible to create a table of values during compilation and simply index into the table during execution. The following example creates an array of the squares of the numbers from 0 to 100 during compilation :

```
DECIMAL   <cr> ok <0>
CREATE  SQUARES  202  ALLOT  <cr> ok <0>

: GEN-SQS
    101  0  DO
        I  I  *                    ( Calculating the square.)
        SQUARES   I  2* +          ( Indexing into the table.)
        W!                         ( Storing the word-length value.)
    LOOP ;  <cr>ok <0>

GEN-SQS   <cr> ok <0>

SQUARES  30  2* + W@  .  <cr> 900 ok <0>      ( Indexing into table.)
```

See also: VALLOT , C, , W, , , , HERE , CREATE

G-65

Dictionary
Management

ALSO

Format:	ALSO <vocabulary-name>
Action:	Specifies that the vocabulary specified by <name> should ALSO be added to the current dictionary search order. The vocabulary will be the first vocabulary searched in the new search order (the "transient" vocabulary). Any subsequent execution of WORDS will display only the words in the transient vocabulary and the words used to specify the search order.
<u>Example:</u>	At the beginning of a program, the words ONLY , ALSO , and DEFINITIONS should be used to specify the search order to be used during compilation (loading) of the program. For example, to allow a program to access words from the FORTH, OS-9, and MATH vocabularies the search order should be set up as follows:

ONLY FORTH <cr> ok <0>   (FORTH is now the ONLY vocabulary searched.)
DEFINITIONS <cr>ok <0>   (Any new definitions will be appended to the
                                                      FORTH vocabulary.)
ALSO OS-9   <cr>ok <0>   (Now OS-9 has been added to the search order.)
ALSO MATH   <cr> ok <0>   (Now MATH has been added to the search order.)

The search order specified above allows all words in the FORTH, OS-9, and MATH vocabularies to be found. Whenever MACH 2 looks for a word it will search the MATH vocabulary first, the MACH vocabulary second and the FORTH vocabulary last. This search order may be visually displayed by using the word ORDER:

ORDER   <cr> ok <0>

Search Order : MATH OS-9 FORTH
Definitions   : FORTH
ok <0>

See also: ONLY , DEFINITIONS

G-66

**Dictionary
Management**

AND   (n1 n2 - n3)

Format: n1 n2 AND

Action: Performs the bit-by-bit logical "AND" of n1 with n2.
                Leaves the result on top of the stack.

Example: The truth table for AND is:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

BINARY <cr> ok <%0>

10101010   <cr> ok <%1>

10001111 AND <cr> ok <%1>

. <cr> 10001010 ok <%0>

DECIMAL <cr> ok <0>

For Assembly Language Programmers:

```
CODE  AND  ( n1  n2 - n3 )
      MOVE.L      (A5)+,D0
      AND.L       D0,(A5)
      RTS
END-CODE
MACH
```

See also: OR , XOR , NOT , BINARY , DECIMAL

G-67

Logic
Operator

ASCII

Format:	ASCII character
Action:	When used inside of a colon definition, ASCII compiles the ascii value of the single character delimited by spaces as a literal. During execution of the definition the ascii value will be put on the stack. When executed immediately, ASCII will leave the ascii value of the character on the stack.

Example: The word GET-COMMAND takes a character from the user and compares it to a list of valid command characters to determine which action the user specified :

```
: GET-COMMAND   ." Next command --> " KEY CR
      CASE
            ASCII L OF ." Turning left..." GOLEFT ENDOF
            ASCII R OF ." Turning right..." GORIGHT ENDOF
            ASCII F OF ." Going forward..." GOFORWARD ENDOF
            ASCII B OF ." Going backwards..." GOBACK ENDOF
      ." Unknown command..."
      ENDCASE  ; <cr> ok <0>

GET-COMMAND   <cr> Next command --> F
Going Forward... ok <0>
```

ASCII helps improve program readability and eliminates the process of looking up character values in an ASCII table. The number formatting process provides a good use for ASCII. The word HOLD requires the ASCII value of the character it should insert into the formatted string on the stack. To print out a number in dollars and cents format :

```
: $ ( n - ) <# # # ASCII . HOLD #S ASCII $ HOLD #> TYPE ;
                   ↑                    ↑
              Inserts the .        Inserts the $

9999 $  <cr> $99.99 ok <0>
```

See also: LITERAL , #

G-68

Character I/O

ASSEMBLER

Format: ONLY ASSEMBLER   or   ALSO ASSEMBLER

Action: ASSEMBLER is the vocabulary which contains all of the
                 assembly language operators.

                 When used with ONLY, ASSEMBLER will become the only
                 vocabulary included in the current search order .

                 When used with ALSO, the ASSEMBLER vocabulary will be
                 appended to the current search order. This will cause
                 ASSEMBLER to become the transient vocabulary (the
                 vocabulary which is searched first). Any other vocabularies in
                 the search order will be searched after the transient vocabulary.
                 WORDS will only display the words in the transient vocabulary
                 and the words used to specify the search order when executed.

Example: To specify a search order in which the ASSEMBLER vocabulary
                 will be searched first and the FORTH vocabulary second :

                 ONLY   FORTH   **<cr>** ok <0>
                 ALSO   ASSEMBLER   **<cr>** ok <0>

                 The word ORDER will display the current search order and
                 the vocabulary to which new definitions are being appended :

                 ORDER   <cr>
                 Search Order : ASSEMBLER FORTH
                 Definitions   : FORTH
                  ok <0>

                 To make the ASSEMBLER vocabulary the only vocabulary which
                 is searched :

                 ONLY   ASSEMBLER   **<cr>** ok <0>

See also: ONLY , ALSO , ORDER , DEFINITIONS

G-69

**Dictionary
Management**

ASSIGNMODULE (a n -)

Format:	" module name" 1+ trap# ASSIGNMODULE

Action: ASSIGNMODULE is used to assign trap modules (specified by name) to trap vectors. Trap modules may only be accessed through one of the 16 software exception trap vectors provided by the 68000 microprocessor. Before a specific trap module may be accessed, ASSIGNMODULE must be used to 'tag' a trap module to a trap number so the system will know which module to call when a trap is executed.

ASSIGNMODULE expects to be passed the address of a C-format string (hence the 1+ above, see ") which contains the module name, and the trap vector to which the module should be 'linked'.

Example: Before we can access the routines in the "Extensions" trap module we must specify which trap vector will be used to access the trap module:

5   CONSTANT   ExtensionsVector   <cr> ok <0>

" Extensions" 1+   ExtensionsVector
ASSIGNMODULE <cr> ok <0>

Now, whenever a TRAP5 is executed, the Extensions trap module will be called.

See also: MAKEMODULE , TCALL

G-70

Application
Word

BASE (- a)

Format:	BASE
Action:	Returns the address of the user variable BASE, which contains the current task's number radix. A task's number base controls all number input/output operations for the task.
Example:	The following format is used to change the number base for a task -
	n BASE !

A stack depth indicator is included with the "ok" message. A punctuation symbol is used in the stack depth indicator to indicate the current number base:

DECIMAL <cr>ok <0>

⎯⎯⎯ No punctuation sign means base 10 (DECIMAL)

BINARY <cr> ok <%0>

⎯⎯⎯ A percent sign (%) means base 2 (BINARY)

HEX <cr> ok <$0>

⎯⎯⎯ A dollar sign ($) means base 16 (HEXADECIMAL)

3 BASE ! (or any base besides those above) <cr> ok <?0>

A question mark (?) means a non-standard base ⎯⎯⎯

Some examples of changing base -

DECIMAL <cr> ok <0>	Set base to base 10 - DECIMAL.
9 <cr> ok <1>	Put a decimal 9 on the stack.
3 BASE ! <cr> ok <?1>	Change base to base 3.
. <cr> 100 ok <?>	9 decimal output when base 3 is in effect, $9_{base\ 10} = 100_{base\ 3}$

See also: DECIMAL , HEX , BINARY

User/Local Variable

G-71

BEGIN

Format: A BEGIN loop has three possible formats:

    BEGIN
    (code to be executed while flag = false)
    flag
    UNTIL

or BEGIN
    (code to be executed each time through loop)
    flag
    WHILE
    (code to be executed while flag = true)
    REPEAT

or BEGIN
    (code to be executed endlessly)
    AGAIN

Action: BEGIN marks the start of either the BEGIN. . . UNTIL, BEGIN
...WHILE...REPEAT, or BEGIN...AGAIN loops. If these loops
repeat, program control will always be rerouted back to
the code which immediately follows the BEGIN.

Example:

    : KEY-CHECK   (Code to be executed
        BEGIN   while flag = false)
            ." No key yet" CR
        ?TERMINAL
        UNTIL ;

(Program
  execution
  will be
  rerouted by
  UNTIL back to
  the code which
  follows BEGIN if
  the flag passed
  to UNTIL is false.)

(?TERMINAL leaves flag
  on stack indicating
  whether or not key has
  been pressed.)

See Also: UNTIL , ?TERMINAL , WHILE , REPEAT , AGAIN

G-72

Program
Control
Structure

BINARY

Format:	BINARY
Action:	Sets the current task's number base - which controls all number input/output operations for the system - to binary. The stack depth indicator will contain a % sign when the system is in the binary base.

Example:

```
DECIMAL  <cr> ok <0>
16  <cr>  ok <1>
BINARY  <cr> ok <%1>
.  <cr> 10000 ok <%0>


DECIMAL  <cr> ok <0>
: NUMBERS  CR  10  0  DO  I  .  LOOP  ;  <cr> ok <0>
BINARY  <cr> ok <%0>
NUMBERS  <cr>

0 1 10 11 100 101 110 111 1000 1001  ok <%0>
```

The definition of BINARY is -

```
DECIMAL  <cr> ok <0>

: BINARY  2  BASE  !  ;  <cr> ok <0>
```

See also: DECIMAL , HEX , BASE

Number
I/O

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

BLK   (- a)

" b-l-k "

Format: BLK

Action: Puts on the stack the address of the system variable whose
                   contents indicate the location of the current input stream
                   (see table below). BLK, >IN, TIB and #TIB are the four system
                   variables responsible for maintaining control of the input
                   stream.

Example: **BLK Value**   **Input Stream**

                   -2   Serial port
                   -1   Text file (loading from a text file)
                   0   Text Input Buffer (TIB)
                   >0   Single block buffer (the number
                                       indicates which block)

CURRENT-INPUT-STREAM will print out which source is being used as the current
input stream when it is executed :

```
:  CURRENT-INPUT-STREAM   BLK  @
     DUP  0>  IF
                 ." Block " .
             ELSE
                CASE
                     -1  OF  ." TextFile"  ENDOF
                      0  OF  ." Text Input Buffer"  ENDOF
                ENDCASE
             THEN
;  ok  <0>
```

CURRENT-INPUT-STREAM   <cr>   Text Input Buffer ok <0>

Since CURRENT-INPUT-STREAM was executed from the keyboard, the text input
buffer was being used as the current input stream. If BLK contains a positive
number, a single block is being loaded and the block number will be printed out.
If BLK contains a negative number or zero, the name of the corresponding input
stream source will be printed. Put the entire example above in a file and try loading
the file as a file and as a single block to generate the other possible messages.

See also: >IN , TIB , #TIB

G-74

System/Local
Variable

BLOCK   (n - a)

Format:	n BLOCK
Action:	BLOCK expects a number on the stack which specifies a certain block (a block is 1024 bytes of data) within the currently open file. If the specified block is not available in memory, BLOCK will read it into the least recently accessed block buffer and put the address of the buffer on the stack. If the specifed block is available in a block buffer in memory, BLOCK will leave the address of that buffer on the stack. The first block in a file is block 0.

Example: In the following example, the address of the first byte of the fourth block in the current file for the system is requested. BLOCK will only read this block in from disk if it determines that the block is not already in a block buffer:

HEX   <cr> ok <$0>

3   BLOCK   <cr> 74FA6 ok <$0>

If BLOCK determines that the requested block is not in memory, BLOCK will figure out which block buffer was the least recently accessed and use that buffer for the next block of information. If that buffer was marked as UPDATE'd, BLOCK will first write the buffer contents out to disk. BLOCK will only read in a block from disk if the block contents are not already in a buffer in memory.

See also: FILEID , BUFFER , LIST

G-75

Storage
I/O

BODY>LINK (a1 - a2)

" body-to-link "

Format:	parameter-field-address BODY>LINK
Action:	BODY>LINK takes the address of the start of the executable code for a word (the PFA), as returned by " ' " or ['] , and returns the address of the link field for the word (the LFA).
<u>Example:</u>	The following sequence may be used to display the dictionary header for the word DUP :

```
'  DUP   BODY>LINK   10   DUMP
```

See also: LINK>BODY , ' , [']

Compilation
Word

BUFFER   (n - a)

Format:	n BUFFER
Action:	Assigns a buffer to block n of the currently open file and leaves the address of the buffer on the stack. BUFFER does not move the contents of the specified block from storage to memory. BUFFER is identical to BLOCK except it does not initially read in the data from disk. The first block in a file is block number 0.

Example: In the following example, a buffer is assigned to block number five in the current file for a task :

HEX <cr> ok <$0>
5 BUFFER <cr> ok <$1>

. <cr> 757A6 ok <$0>

Setting the Current File

The FORTH disk/storage I/O words all act upon the current file. The current file is the file whose word-length (16-bits) path number is stored in the system variable FILEID. Each time a file is successfully opened, a path number which uniquely identifies the file is returned. To make the file the current file, this path number should be stored into the FILEID system variable. BLOCK, BUFFER, LIST, and LOAD use FILEID to determine which file to access. An example of making a file current is given below:

ALSO OS-9 <cr> ok <0>
" MyFile" 1+ $OPEN . <cr> 4 ok <0>   (Open the file MyFile.
                                                                                          $OPEN will return a path number.)
4 FILEID W! <cr> ok <0>   (Store this path number in FILEID.
                                                                                          Now MyFile is the current file.)

See also: BLOCK

Storage
I/O

BYE

Format:	BYE
Action:	Used to leave MACH2 and return to the OS-9 shell. Restores the previous terminal characteristics and uses F$Exit to terminate the MACH2 process.

Assembly Note: This is how F$Exit could be called from assembly language:

```
CODE ByeBye (  -  )
    MOVEQ.W  #0,D1     \ status code to return to parent
    OS9      F$Exit    \ 'call' F$Exit
END-CODE
```

See also: $

OS-9
Interface

C!   (c a -)

" character-store "

Format: byte-value address C!

Action: Stores the byte-length (8-bit) value at the specified address. C! is referred to as "character-store" because it is often used to manipulate ascii characters (which are expressed as byte-length values).

Example: The following example shows that even though numbers placed on the stack are represented using 32-bits, the operator C! will take only the least significant byte (8-bits) and store it in memory :

HEX   <cr> ok <$0>
VARIABLE STORAGE   <cr> ok <$0>

2F STORAGE C! <cr> ok <$0>

The number 2F as it appears on the stack.

0 0	0 0	0 0	2 F
1 byte	1 byte	1 byte	

Bytes in memory :

lower memory -> higher memory

STORAGE

C! takes only the lowest-order byte from the stack and stores it starting at the specified address.

For Assembly Language Programmers:

```
CODE C! ( c  a  -  )
     MOVE.L    (A5)+,A0
     MOVE.L    (A5)+,D1
     MOVE.B    D1,(A0)
     RTS
END-CODE
MACH
```

See also: C@ , W! , W@ , ! , @

G-79

Memory
Operator

C ,   (c -)

" c-comma "

Format:	8-bit-value C,
Action:	Lays the 8-bit value into the next available memory location in the dictionary. The pointer to the next available dictionary location is incremented by 1 byte.

Example:

In the following example, a table of data is being created in the dictionary. CREATE is used to make the dictionary header (only the contents of a definition are kept in the dictionary, the headers are kept in a separate location). Next, C, is used to lay the desired data into the dictionary. Each time C, is used, the HERE pointer (the pointer to the next available dictionary location) is incremented by 1 byte. When TABLE is executed, it will push the address of the first byte of data, the A, on the stack :

```
CREATE  TABLE  <cr> ok <0>

ASCII  A  C,  <cr>ok <0>
ASCII  B  C,  <cr> ok <0>
ASCII  C  C,  <cr> ok <0>

TABLE  C@  EMIT  <cr>  A ok <0>
TABLE  2+  C@  EMIT  <cr>  C ok <0>
```

NOTE:

It is not necessary to keep track of address boundaries when using C, since all other words which affect the HERE pointer (the pointer to the next available dictionary location) will adjust the pointer to an even boundary before preceding.

See also: , , W, , ALLOT , HERE

G-80

Compilation Word

C@ (a - c)

" c-fetch "

Format:	address C@
Action:	Replaces the address on top of the stack by the 8-bit value which is stored at the address. The upper 3 bytes of the 4 byte value returned are set to zero.

Example:

The following example shows that the C@ operator will return the 8-bit value which is stored starting as the specified address :

HEX   <cr> ok <$0>
VARIABLE   STORAGE   <cr> ok <$0>
1F   STORAGE   C!   <cr> ok <$0>

Bytes in memory :

↓STORAGE

lower memory -> higher memory

0 0	0 0	0 0	1 F

STORAGE   C@   .   <cr> 1F ok <$0>

For Assembly Language Programmers:

```
     CODE  C@  ( a - c )
            MOVE.L    (A5),A0
            CLR.L     D0
            MOVE.B    (A0),D0
            MOVE.L    D0,(A5)
            RTS
     END-CODE
     MACH
```

See also: C!

G-81

Memory Operator

CASE

Format:	CASE is used in the following format:
	
	CASE
	n OF (code executed if n is matched) ENDOF
	n1 OF (code executed if n1 is matched) ENDOF
	n2 OF (code executed if n2 is matched) ENDOF
	            •
	            •
	nn OF (code executed if nn is matched) ENDOF
	(code executed if no match was made above)
	ENDCASE

Action:	CASE marks the beginning of the CASE...OF...ENDOF...ENDCASE program control structure (also referred to as a CASE statement). The number on the stack is compared to each number preceding an OF...ENDOF pair until either a match is found or until there are no more OF..ENDOF pairs left. If a match is made, the number will be dropped from the stack and the code between the corresponding OF and ENDOF will be executed. Upon reaching the ENDOF, program control will be redirected to the code which immediately follows the ENDCASE. If no match is made, any code between the last ENDOF and the ENDCASE will be executed. After this, ENDCASE executes. Its function is to drop the top number-the unmatched number-from the stack. Any number of OF...ENDOF pairs may be used within a CASE statement.

See also: OF , ENDOF , ENDCASE

G-82

Program
Control
Structure

CMOVE   (a1 a2 n -)

" c-move "

Format:	source-addr destination-addr #of-bytes CMOVE
Action:	Moves n bytes located starting at the source address to memory starting at the destination address. The move begins with the first byte in the source string and continues until the last byte in the source string has been moved. CMOVE is good for moving strings down toward lower memory locations.
Example:	The following example creates an array of ascii characters in the variable space (using VARIABLE and VALLOT), and then shifts the whole array 12 bytes downward in memory.

```
VARIABLE  DEST     8 VALLOT<cr> ok <0>
VARIABLE  SOURCE    8  VALLOT  <cr> ok <0>

SOURCE  12  EXPECT  <cr>MACH 2<cr> ok <0>

SOURCE   DEST   12  CMOVE  <cr> ok <0>
DEST  10  DUMP  <cr>
0765DE:  4D41 4348 2031 0DFF FFFF FFFF FFFF FFFF   MACH 2 .ÛÛÛÛÛÛÛÛ

ok <0>
```

See also: CMOVE> , DUMP , VALLOT

G-83

Memory
Operator

CMOVE>   (a1 a2 u -)
" c-move-up "

Format:	source-addr destination-addr #of-bytes CMOVE>
Action:	Copies a region of memory n bytes long, beginning at the source address, to memory beginning at the destination address. The move starts by moving the last byte in the source string to the last position in the destination string and continues until the first byte in the source string has been reached and moved. This version of CMOVE is usually used for sliding strings toward higher memory locations.
Example:	The following example creates a string of ascii characters in memory and then shifts the entire string 2 bytes toward higher memory.

VARIABLE SOURCE 4 VALLOT <cr> ok <0>

SOURCE 8 EXPECT <HELLO<cr> ok <0>

SOURCE 10 DUMP <cr>
0765DE: 4845 4C4C 4F00 FFFF FFFF FFFF FFFF FFFF   HELLO . ÙÙÙÙÙÙÙÙÙÙ
ok <0 >

SOURCE SOURCE 2+ 8 CMOVE> <cr> ok <0>

SOURCE 10 DUMP <cr>
0765DE: 4845 4845 4C4C 4F00 FFFF FFFF FFFF FFFF   HEHELLO . ÙÙÙÙÙÙÙÙ
ok <0>

If CMOVE had been used instead of CMOVE> the results would have been much different. Because CMOVE starts moving bytes from the beginning of the string, when it is used to shift bytes up towards higher memory locations, it will cause this overwrite error :

SOURCE SOURCE 2+ 8 CMOVE <cr> ok <0>

SOURCE 10 DUMP <cr>
0765DE: 4845 4845 4845 4845 4845 FFFF FFFF FFFF FFFF   HEHEHEHEHEÙÙÙÙÙÙ
ok <0>

See also: CMOVE , FILL

G-84

Memory
Operator

CODE

Format:	CODE <name-of-routine>
	...assembly language...
	END-CODE

Action:

Defining word, similar to : , which is used to create new
dictionary entries. The compile-time action of CODE involves
creating a new header for the name which follows it, putting
the system into the compiling state, adding the ASSEMBLER
to the dictionary search order and setting the smudge bit of
the new definition so that the definition will not be visible until
completed.

The opcodes for the assembly language words used between
CODE and END-CODE will be compiled into the new definition.

Example:

CODE is used below to create the word WIPEOUT. If run,
WIPEOUT will destroy large portions of the current contents
of the computer's memory so please, do NOT try this at
home ! Notice that the MACH 2 assembler allows normal,
assembly language branches to occur within CODE definitions.

```
CODE  WipeOut              (New word WIPEOUT created.)
@1      CLR.L    (A0)+     (Assembler vocabulary now included search
        BRA.S   @1          order.)
END-CODE                   (CODE definitions end with END-CODE.)
```

For Advanced Programmers:

Note that the mnemonics in the assembler vocabulary are all immediate words
which compile their corresponding opcodes into the definition being created.

See also: END-CODE , ;CODE , MACH, :

G-85

Compilation
Word

COMPILE

Format:	Generally used in the format -

: <forth-definition>
　　COMPILE run-time-word
　　. . . code to be executed during compilation . . . ;
IMMEDIATE

Action: Although <forth-definition> is an immediate word,
COMPILE will force the word following it
(<run-time-word>) to be compiled, instead of run, when
<forth-definition> executes. When <forth-definition> is
later used within another definition, only <run-time-word> will be compiled into the definition, the other
words in <forth-definition> will run immediately and
will not generate any compiled code. This type of word
is known as a compiling word. When compiling words
are used in the creation of a new definition they perform
compile-time actions and may compile run-time code
into the new word. In <forth-definition>, <run-time-word>
is the run-time code which would be compiled into a new
word and "code to be executed during compilation" is the
compile-time actions which would be performed.

Example: DO is an example of a FORTH compiling word which does
add run-time code to the word being defined (this is a
simplified version of DO) :

```
: 2>R  >R  >R  ;   <cr> ok <0>
: DO  COMPILE  2>R  HERE  ;  IMMEDIATE <cr> ok <0>
```

run-time code
which is compiled
in the definition
which contains DO

code executed
during compilation

compiling words
are always
immediate

During compilation of a new definition, DO must leave the current address
on the stack so LOOP will know how far back it will have to jump if looping
is ever required. All that DO will leave in the final compiled version of the
new word is the run-time action of DO which is to move two numbers
from the parameter stack to the loop stack.

See also: [COMPILE] , IMMEDIATE

Compilation
Word

CONSTANT (n -)

Format: 32-bit-value CONSTANT <name>

Action: Defining word which creates a dictionary entry using <name>
                     and stores the 32-bit-value in its parameter field. The
                     run-time action of words created by CONSTANT is to push
                     the value in their parameter field on the stack.

Example: A high-level definition of CONSTANT could be:

                     : CONSTANT (n -) CREATE , DOES> @ ;

5280 CONSTANT FT/MILE <cr> ok <0>

FT/MILE . <cr> 5280 ok <0>

In MACH 2 constants and variables are state dependent words. The above
definitions of is a simplified version of the actual CONSTANT definition.

See also: VARIABLE , CREATE , DOES>

G-87

Defining
Word

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

CONVERT   (n1 a1 - n2 a2)

Format: seed address-of-string-to-be-converted CONVERT

Action: CONVERTs the ascii string at the specified address to a
                 binary number. CONVERT expects on the parameter stack a
                 single-length (32-bit) number and the address of the string.
                 The string conversion process begins with the second byte in
                 the string, the first byte (assumed to be a length byte) is skipped.
                 CONVERT continues working on the string, and accumulating
                 the converted number into the seed on the stack, until it
                 encounters an ascii value which does not represent a valid digit
                 in the current number base. CONVERT leaves on the stack the
                 single-length value of the number it has CONVERTed so far and
                 the address of the first invalid ascii value it encountered.

Example: In the following example, a string to be CONVERTed to a number
                 is stored into a variable location using EXPECT. Then CONVERT is
                 used to convert the string to a number :

DECIMAL <cr> ok <0>   (CONVERT converts the string to a number with regard to
                             the current number base.)

VARIABLE STORAGE <cr> ok <0>   (The string will be stored in the variable
4 VALLOT <cr> ok <0>   STORAGE. 4 extra bytes have been
                                     allocated for STORAGE so a string which
                                     contains up to 8 bytes may be kept in
                                     STORAGE.)

STORAGE 1+ 8 EXPECT <cr> 432-4000 <cr> ok <0>
                             (EXPECT will start storing the string in the second byte of
                               STORAGE since CONVERT skips past the first byte in the
                               string.)

0 STORAGE CONVERT <cr> ok <2>   (The address on top of the stack points to)
                                   (the first unconvertable character)
.S <cr> 432 441468 <-- TOP ok <2> (CONVERT encountered. This should be the)
                                   ('-'. The second number is the value of the)
                                   (number CONVERT has managed to convert so)
C@ EMIT <cr> - ok <1>   (far. If the number passed to CONVERT had not)
                                   (been a zero, the CONVERT would have)
           ↑   (appended the 432 to the end of the original)
                                   (number--i.e. if the number had been a 98,)
(The first unconvertable character (CONVERT would have returned a 98432.)
CONVERT encountered was the
minus sign.)

See also: EXPECT , VARIABLE , VALLOT , NUMBER?

G-88

Number
I/O

COUNT (a - a n)

Format:	address-of-length-byte-of-string COUNT
Action:	Given the address of the length byte of a string in memory, COUNT will push the address where the text of the string actually begins (old address+1 to skip over length byte) and the length of the string on the stack. COUNT is particularly useful for setting a string up for TYPE since the stack after COUNT holds the two values TYPE expects.

Example: " returns the address of a string in memory :

" hello " \<cr\> ok \<1\>
COUNT TYPE \<cr\> hello ok \<0\>

In this example the stack is displayed after COUNT to show the length and address left by COUNT :

" HI !" \<cr\> ok \<1\>

COUNT \<cr\> ok \<2\>

.S \<cr\> 17795 4 \<- TOP ok \<2\>

TYPE \<cr\> Hi ! ok \<0\>

See also: TYPE , "

Character
Output

COUNTER

Format: COUNTER

Action: Zeros a variable to be used for keeping track of time
                 and initiates the start of a timing process. The clock
                 rate (the number of ticks per second) is usually 100
                 hertz.

Example: The following example shows how COUNTER...TIMER may
                 be used to time a null loop:

```
:  NULL-LOOP
     COUNTER
     10000  0  DO  LOOP
     TIMER   ;  <cr> ok <0>

NULL-LOOP   <cr> 2 ticks | 100 hertz ok <0>
```

                 Note that the time returned by COUNTER...TIMER is
                 clock rate dependent.

See also: TIMER

FORTH
Tool

CR

Format:	CR
Action:	Prints a carriage return and linefeed.

Example: The following word prints "hello !" on the screen 10 times. A CR is inserted once each time though the loop so that each "hello !" will be on its own line -

`: HELLO  CR 10  0  DO  ." hello !"  CR  LOOP  ;  <cr> ok <0>`

`HELLO  <cr>`

```
hello !
hello !
hello !
hello !
hello !
hello !
hello !
hello !
hello !
hello ! ok <0>
```

See also: EMIT

G-91

Character
I/O

CREATE

Format:	CREATE <name>
Action:	Defining word which builds a header for <name> in the names space.

Example: CREATE is commonly used to build defining words.
An example of a useful defining word is the word ARRAY
which defines a two-dimensional array :

```
:  INDEX-ARRAY { x y addr | cols index -- a }
      addr @ -> cols
      cols y * -> index
      x addr + index + 4 + ;

:  ARRAY ( rows cols -- )
      CREATE
         cols ,
           rows cols * ALLOT
      DOES>
         INDEX-ARRAY ;
```

2 3 ARRAY HOLLYWOOD-SQUARE   (Define a 2x3 array.)

1 2 HOLLYWOOD-SQUARE C@   (Index into array.)

See also: HERE , ALLOT , VARIABLE , VALLOT

Compilation Word

D+   (d1 d2 - d3)

" d-plus "

Format: d1 d2 D+

Action: Adds two 64-bit numbers and leaves the 64-bit
                 sum on the stack.

See also: D< , DNEGATE

G-93

Arithmetic Operator

D<   (d1 d2 - f)

" d-less-than "

Format: d1 d2 D<

Action: Compares the two double-length (64-bit) numbers
on top of the stack. Returns a true (non-zero) flag if
d1 is less than d2 and a false (zero) flag if d1 is not
less than d2.

See also: D+ , DNEGATE

G-94

**Arithmetic
Operator**

DEBUG

Format:	DEBUG or DEBUG <name-of-FORTH-word>
Action:	Used to invoke the debugger. If DEBUG is executed immediately it will cause the debugger to be immediately entered. If DEBUG is executed immediately with the name of a FORTH word following it a breakpoint will be set at the first instruction in the word. The next time the word is executed, the debugger will be entered. DEBUG may also be compiled into a definition.

Example: Three ways to set a breakpoint at the first instruction in the
word FRED: **: FRED DUP . ;**

#1. Interactively install a breakpoint at the first instruction of FRED:

DEBUG   FRED   <cr> ok <0>
FRED   <cr>

00A544:   MOVE.L   (A5),-(A5)   DUP
>

The next time FRED is executed the debugger will stop execution and display
the first instruction in FRED.

#2   Compile DEBUG into the FRED definition:

: FRED DEBUG DUP ; <cr> ok <0>
FRED   <cr>

00A544:   MOVE.L   (A5),-(A5)   DUP
>

The next time FRED is executed the debugger will stop execution and display
the first instruction in FRED.

#3   Interactively execute DEBUG and set a breakpoint at FRED:

DEBUG   <cr>

FF4E72:   TST.L   (A5)
> BR   FRED <cr>

ASSEMBLER
Word

DEFINITIONS

Format: <vocabulary name> DEFINITIONS

Action: Makes the transient vocabulary (the vocabulary which is
                           currently being searched first in all dictionary searches) the
                           vocabulary to which all subsequent definitions will be
                           appended.

Example: If DEFINITIONS is used after ONLY, it will make the only
                           vocabulary being searched the vocabulary to which any
                           new definitions will be appended :

ONLY FORTH <cr>ok <0>   (At this point, ONLY the FORTH vocabulary is being
                                searched which means the FORTH vocabulary must
                                be the vocabulary which is being searched first.)

DEFINITIONS <cr>ok <0>   (DEFINITIONS will see that the FORTH vocabulary
                                is being searched first and will make the FORTH
                                vocabulary the vocabulary to which all new
                                definitions will be appended.)

If DEFINITIONS is used after ALSO, it will make the vocabulary whose name was just
passed to ALSO the vocabulary to which new definitions will be appended :

ALSO ASSEMBLER <cr>ok <0>   (This use of ALSO will make ASSEMBLER the
                                    new transient vocabulary.)

DEFINITIONS <cr> ok <0>   (Since the ASSEMBLER vocabulary is now
                                    the vocabulary being searched first,
                                    DEFINITIONS will make any subsequent
                                    definitions be appended to the ASSEMBLER
                                    vocabulary.)

The word ORDER will display the vocabulary to which new definitions are currently being
appended :

ORDER <cr>
Search Order   : ASSEMBLER FORTH
New Definitions : ASSEMBLER
 ok <0>

See also: ONLY , ALSO , ORDER

G-97

**Dictionary
Management**

DEPTH   (- n)

Format:	DEPTH
Action:	Returns a count of the number of items on the parameter stack.

Example:

10 20 30 40 50 60 70 <cr> ok <7>
DEPTH . <cr> 7 ok <7>

The depth is 7 since there are
seven items on the stack.

See also: .S

FORTH
Tool

DISK   (- a)

Format:	DISK
Action:	Puts on the stack the address of a variable which holds the file system result code for the most recent file access.

Example:

The following definition culd be used to check the result of BLOCK:

```
:  DISK-ERROR?
     DISK  W@
     ?DUP
     IF
         CR  ." Disk Error" .
     THEN
;
```

System/User Variable

DNEGATE (d - -d)

" d-negate "

Format: d DNEGATE

Action: Changes the sign of the double-length (64-bit)
               number on top of the stack (two's complement).
               The negation of d is d = 0 - d .   " d-negate "

See also: D< , D+

G-100

Arithmetic
Operator

DO (n1 n2 -)

Format: There are two formats for DO loops:

    limit index
    DO
      (code to be executed each time through loop)
    LOOP

or limit index
    DO
      (code to be executed each time through loop)
    increment value
    +LOOP

Action: DO marks the beginning of the DO. . .LOOP or DO. . .+LOOP program control structures. DO loops are used when a certain sequence of operations must be executed a known number of times. The two numbers passed to DO are called the limit (n1) and index (n2) and are used by either LOOP or +LOOP to determine the number of times the code within the loop should be executed. DO is executed only once and its action is to move the limit and index to the loop stack for LOOP or +LOOP.

Example: : NUMBERS 5 0 DO I . LOOP ; <cr> ok <0>

NUMBERS <cr> 0 1 2 3 4 ok <0>

The effect of DO -

Parameter and loop stack right before DO:

Index 0
Limit 5
Param Loop

Parameter and loop stack right after DO:

0 Index
5 Limit
Param Loop

See Also: LOOP , +LOOP , I , J

G-101

Program Control Structure

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z {|}~

DOES> (- a) run-time (-) compiling

"does"

Format: : <new defining word>
                   (compile-time behavior)
                   DOES>
                   (run-time behavior)
               ;

Action: Used in the definition of a new defining word to define the run-time
              action of words created by the defining word. The words which follow
              DOES> in a defining word definition will be executed by words
              created by the defining word, not by the defining word itself. When
              a word created by a defining word which uses DOES> is executed, its
              parameter field address is pushed on the stack and then any words
              following DOES> are executed.

Example: The defining word CONSTANT could be defined as follows:

              : CONSTANT (n -)

                  CREATE   (Creates a new dictionary entry using
                                    the next name in the input stream.)

                  ,   (Stores the constants value in its
                                    own parameter field.)

                  DOES>   (Marks the end of the compile-time
                                    behavior and the beginning of the
                                    run-time behavior.)

                  @   (When a word created using CONSTANT
                                    is executed DOES> will push the word's
                                    parameter field address on the stack.
                                    @ gets the value which was stored in
                                    parameter field during compilation and
                                    puts it on the stack.)

              ;

12 CONSTANT DOZEN <cr> ok <0>
DOZEN . <cr>12 ok <0>

See also: CREATE

G-102

Compilation
Word

DROP   (n -)

Format: n DROP

Action: Removes the top item on the stack.

Example: 3 2 DROP <cr> ok <1>

For Assembly Language Programmers:

```
CODE  DROP  ( n - )
      ADDQ.L   #4,A5
      RTS
END-CODE
MACH
```

See also: DUP , OVER , PICK

Stack
Manipulation

DUMP   (a n -)

Format:	address count DUMP
Action:	Displays the contents of n memory locations starting at the specified address. The contents of memory locations are always displayed as hexadecimal ascii values in groups of 16 bytes.

Example: It is convenient to use DUMP for examining string data in memory:

```
:  GreetString  ( - )    "  Hello !! "  ;  <cr> ok <0>
'  GreetString  10  DUMP  <cr>

        2 3   4 5   6 7   8 9   A B   C D   E F   0 1     23456789ABCDEF01
055192:  601A  0848  656C  6C6F  2021  2100  2B1F  4E75   a..Hello.!!.+.Nu
ok <0>
```

G-104

FORTH
Tool

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

DUP   (n - n n)
" dupe "

Format: n DUP

Action: Duplicates the number on top of the stack.

Example: 6 DUP <cr> ok <2>

               .S   <cr> 6 6 <- TOP ok <2>

For Assembly Language Programmers:

```
     CODE  DUP ( n - n n )
           MOVE.L    (A5),-(A5)
           RTS
     END-CODE
     MACH
```

See also: DROP , OVER , PICK , ?DUP

G-105

Stack
Manipulation

ELSE

Format:
```
flag
IF
  ( code executed if flag passed to IF is true )
ELSE
  ( code executed if flag passed to IF is false )
THEN
( code which is always executed )
```

Action:
ELSE is the middle word in the IF. . .ELSE. . .THEN program control structure. The code between the ELSE and THEN will be executed if a false flag (zero) is passed to IF. Once the THEN is reached, program execution will continue on to the code following the THEN. If a true flag (non-zero) is passed to IF, the code between the IF and ELSE will be executed. Upon reaching the ELSE, program execution will be redirected to the code following the THEN.

Example:
FAREWELL expects a number on the stack. If the number is greater than 3, a "good day" message will be issued. If the number is less than 3, a "bad day" message will be issued :

```
3  CONSTANT  GOOD  <cr> ok <0>
: FAREWELL  ( n - )
    ." Have a "
    GOOD  >
    IF
      ." good "          ( Code executed if flag
    ELSE                   passed to IF is true. )
      ." bad "           ( Code executed if flag
    THEN                   passed to IF is false. )
    ." day ! "  ;        ( The code following
                           the THEN is always
                           executed. )
```

(The greater-than comparison operator is used to generate a flag for IF.)

2 FAREWELL <cr> Have a bad day ! ok <0>

5 FAREWELL <cr> Have a good day ! ok <0>

See Also: IF , THEN

G-106

Program
Control
Structure

EMIT (n -)

Format: ascii-value EMIT

Action: Prints the character whose ASCII value is on the stack.

Example: In the following example ASCII is used to put an ascii value on
                          the stack and EMIT is used to type the character represented
                          by the ascii value out to the current output device:

ASCII A EMIT <cr>A ok <0>

In this example, the ascii value for an "M" is put on the stack before EMIT is executed :

HEX   <cr> ok <$0>

4D EMIT <cr> M ok <$0>

EMIT is also used in the definitions of CR and SPACE :

HEX <cr>ok <$0>

: SPACE 20 EMIT ; <cr> ok <$0>

: CR 0D EMIT ; <cr> ok <$0>

See also: CR , SPACE

G-107

Character
I/O

EMPTY

Format:	EMPTY
Action:	Deletes every dictionary entry which is not a kernel word. EMPTY performs the same special features as FORGET when deleting certain types of words. EMPTY is usually used to return the system to its start-up state.

WARNING:

Breakpoints set using the MACH2 or OS-9 debugger become invalid when the definition in which the breakpoint was originally set is removed from the dictionary with the use of EMPTY or FORGET. When a breakpoint is set, the original instruction at the breakpoint address is replaced with a special opcode. The original instruction is saved away by the debugger. The next time the breakpoint address is executed, or when the breakpoint is cleared, the old instruction will be swapped back into the code. If the original definition was removed from the system and a new definition loaded in, the debugger will try to replace sections of code in the new definition with old instructions which it saved away previously.

To avoid having new definitions damaged, clear all breakpoints immediately before or after the use of EMPTY or FORGET.

The MACH2 debugger will print an error message if the breakpoint conflict described above is detected.

See also: FORGET

G-108

Dictionary Management

EMPTY-BUFFERS

Format: EMPTY-BUFFERS

Action: Clears the update bits of each block buffer so that the contents of
                 the buffers will not be saved to storage.

See Also: UPDATE , FLUSH

G-109

Storage
I/O

END-CODE

Format: CODE name-of-routine   ...assembly language... END-CODE

Action: Used to end CODE definitions. It restores the base, cuts the
               first vocabulary from the search order (which should be the
               ASSEMBLER) and then leaves compile-mode. Note that
               END-CODE does not lay in an RTS ($4E75) as ';' does. This
               must be explicitly done in a code definitions.

Example: END-CODE concludes the definition of MAX by snipping the
               assembler from the dictionary search and then leaving the
               compiling mode.

```
CODE MAX      ( n1  n2  -  n3 )
     MOVE.L      (A5)+,D0
     CMP.L       (A5),D0
     BLE.S       @1
     MOVE.L      D0,(A5)
@1 RTS                        ( Include the RTS )
END-CODE                      ( Leave compile mode )
```

See also: CODE , ;CODE , MACH

G-110

Compilation
Word

ENDCASE   (n -)

Format:	ENDCASE is used in the following format:

```
CASE
n   OF  ( code executed if n is matched )  ENDOF
n1  OF  ( code executed if n1 is matched )  ENDOF
n2  OF  ( code executed if n2 is matched )  ENDOF
                        •
                        •
nn  OF  ( code executed if nn is matched )  ENDOF
( code executed if no match was made above )
ENDCASE
```

Action: ENDCASE is the terminating word in the CASE...OF... ENDOF...ENDCASE program control structure. If a match is made within the CASE statement, program execution is routed to the code which immediately follows the ENDCASE.

Example:

```
: TEST ( n - )  CASE
                    1  OF ." ONE" ENDOF
                    2  OF ." TWO" ENDOF
                    3  OF ." THREE" ENDOF
                  " No match."
                ENDCASE
;  <cr> ok <0>

1  TEST  <cr> ONE ok <0>

4  TEST  <cr> No match. ok <0>
```

See also: CASE , OF , ENDOF

**Program
Control
Structure**

ENDOF

Format: ENDOF is used in the following format:

                      CASE
                      n   OF (code executed if n is matched) ENDOF
                      n1 OF (code executed if n1 is matched) ENDOF
                      n2 OF (code executed if n2 is matched) ENDOF
                                    •
                                    •
                      nn OF (code executed if nn is matched) ENDOF
                      (code executed if no match was made above)
                      ENDCASE

Action: OF is the ending word in the OF...ENDOF structure used
                      within a CASE statement. If the number preceding an
                      ENDOF's corresponding OF is matched, the code between
                      the OF and ENDOF will be executed.

See also: CASE, OF, ENDCASE

G-112

Program
Control
Structure

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

EXECUTE   (a -)

Format:	parameter-field-address EXECUTE
Action:	EXECUTE executes the word whose parameter field address is on the stack. It does this by moving the parameter field address to the subroutine stack and executing an RTS. This causes the word to be executed. EXECUTE has the same effect as just executing the word itself by typing its name.
Example:	The word ' will put the parameter field address of the word which immediately follows it on the stack. The parameter field address is the address where the executable code for a word begins. EXECUTE will take the PFA of a word and execute it just as if the name of the word had been typed :

```
ONLY  FORTH   <cr> ok <0>

'  ORDER  EXECUTE   <cr>
Search Order  :  FORTH
Definitions     :  FORTH
  ok  <0>
```

The word ORDER was "ticked" to get its
pfa and then executed by EXECUTE.

See also: ' , [']

Program
Control
Structure

G-113

EXIT

Format:	EXIT
Action:	Prematurely exits the definition it is in by removing the address on top of the subroutine stack and jumping to the address which was second on the subroutine stack. This is equivalent to inserting a ';' in the middle of the definition.

If an EXIT is to be used within a DO loop, the limit and index should be removed from the loop stack before the EXIT is executed so that the loop stack is left in the same state it was in before the DO loop started execution.

Example: The EXIT in DANGLING acts as if it were the ; and causes execution of the word to terminate :

```
: DANGLING
      ." This sentence will not be" EXIT ." finished." ; <cr> ok <0>

DANGLING  <cr> This sentence will not be  ok <0>
```

The following example shows how EXIT may be used to force an exit from a
PEGIN...AGAIN loop :

```
: BACKDOOR  BEGIN ." Hello" ?TERMINAL IF  EXIT THEN  AGAIN ; <cr>ok <0>
BACKDOOR  <cr> Hello

Hello

Hello

Hello <cr> ok <0>
```

See also: LEAVE

G-114

Program
Control
Structure

EXPECT   (a n -)

Format:	dest-addr number-of-chars EXPECT
Action:	Waits for n characters to be entered from the keyboard, transferring each one to memory starting at the specified address. A carriage return will cause EXPECT to terminate before n characters have been received. Otherwise, EXPECT will keep waiting until all n characters have been received. The system variable SPAN contains the number of characters actually received during the most recent execution of EXPECT.
Example:	The word CAPTURE will take characters received by an execution of EXPECT and create a counted string. A counted string is a string which holds its length in the first byte (the length byte). The characters of the string immediately follow the length byte. Counted strings are a convenient format for operators such as COUNT and TYPE :

```
VARIABLE  TEMP  20  VALLOT  <cr> ok <0>
: CAPTURE
        TEMP  1+                  ( Putting an address on the stack for EXPECT.
                                    The first byte in the TEMP storage area is
                                    skipped over because this is where the length
                                    byte will be inserted.)
        #19  EXPECT               ( EXPECT will wait for 19 characters or a carriage
                                    return, whichever occurs first.)
        SPAN  @  TEMP  C!         ( Immediately after EXPECT is executed, SPAN
                                    will contain the number of characters received
                                    by EXPECT. This number will be stored in the
                                    first byte of the TEMP storage area to form a
                                    length byte for the string.)
        TEMP  COUNT  TYPE        ( This line will print out the string stored in TEMP.)
; <cr> ok <0>
CAPTURE  <cr>  Hello there.<cr> Hello there. ok <0>
```

See also: SPAN , -TRAILING , COUNT , TYPE , NUMBER?

Character
I/O

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

FILL   (a n b -)

Format: address number-of-bytes-to-fill fill-character FILL

Action: Fills n bytes of memory starting at the specified address
                 with the specified byte value.

Example: In the following example a 10 byte scratch area is created
                 in memory. Then, the first 7 bytes of the scratch area are
                 filled with the ascii value for a "C". The effect of the FILL
                 operation is verified by typing out the contents of the
                 SCRATCH location:

                 CREATE SCRATCH 10 ALLOT <cr> ok <0>

                 SCRATCH 7 ASCII C FILL <cr> ok <0>

                 SCRATCH 10 TYPE <cr> CCCCCCCT Ù ok <0>

See also: TYPE , ASCII

G-116

Memory
Operator

FIND   (a - a n)

Format:	address-of-name-string FIND
Action:	Searches the dictionary, using the current dictionary search order, for the name specifed in the string whose address is on the stack. The first byte in the name string is the length byte. If the search is successful, FIND returns the link field address of the word and a true (non-zero) flag. The flag is a1 if the word found is IMMEDIATE and a -1 if the word is not IMMEDIATE. If the search is unsuccessful, FIND leaves the original address of the name string and a false (0) flag.

Example: The following stack notations show the three possible
FIND results :

```
( address  -  address 0 )      NOT FOUND
( address  -  LFA  -1 )         FOUND, NOT IMMEDIATE
( address  -  LFA  1 )          FOUND , IMMEDIATE
```

EXIST? will take the next word from the input stream and will use FIND to determine
if the word exists in the dictionary :

```
: EXIST?
      32  WORD          ( Get the next word from the input stream.)
      FIND             ( Try to FIND it. )
      IF
         ." Yes"       ( If FIND returns a 1 or -1 on top of the stack
      ELSE               the word does exist. )
         ." No"        ( If FIND returns a 0 on top of the stack the
      THEN               word does not exist. )
      DROP             ( DROP the address returned by FIND. )
;
```

NOTE: MACH 2 is a subroutine-threaded implementation. The conventional CFA
(code field address) does not exist. The FORTH-83 standard requires FIND to
return a CFA. The MACH 2 FIND will return the LFA instead.

See also: ' , ['] , IMMEDIATE , ?INCLUDE"

Compilation
Word

FLUSH

Format:	FLUSH
Action:	Frees up all the block buffers. Writes all block buffers which are marked as updated to disk and then marks all the buffers as unmodified and available. SAVE-BUFFERS is used when the block buffers need to be saved to disk.

FLUSH is called when BYE is executed.

See Also: UPDATE , SAVE-BUFFERS , EMPTY-BUFFERS

G-118

Storage
I/O

FORGET

Format: FORGET <name>

Action: Searches for <name> using the current dictionary search. If <name> is found, <name> and all words added to the dictionary after name, regardless of vocabulary, are deleted from the dictionary. If <name> is not found an error message is issued.

FORGET performs the following actions:

1.   Reclaims the code space for all forgotten definitions.

2.   Reclaims the names space corresponding to all forgotten definitions.

3.   Reclaims the variable space corresponding to all forgotten variables.

4.   Removes any forgotten vocabularies from the list of known vocabularies (all words in the vocabulary are also forgotten.

See also: EMPTY

G-119

**Dictionary
Management**

FORTH

Format:	ONLY FORTH or ALSO FORTH
Action:	FORTH is the vocabulary which contains all words which make up the FORTH kernel.

When used with ONLY, FORTH will become the only vocabulary included in the current search order .

When used with ALSO, the FORTH vocabulary will be appended to the current search order. This will cause FORTH to become the transient vocabulary (the vocabulary which is searched first). Any other vocabularies in the search order will be searched after the transient vocabulary. WORDS will only display the words in the transient vocabulary and the words used to specify the search order when executed.

Example:

To specify a search order in which the ASSEMBLER vocabulary will be searched first and the FORTH vocabulary second :

ONLY   FORTH   <cr>ok <0>
ALSO   ASSEMBLER   <cr> ok <0>

The word ORDER will display the current search order and the vocabulary to which new definitions are being appended :

ORDER   <cr>
Search Order : ASSEMBLER FORTH
Definitions   : FORTH
 ok <0>

To make the FORTH vocabulary the only vocabulary which is searched :

ONLY   FORTH   <cr> ok <0>

See also: ONLY , ALSO , ORDER , DEFINITIONS

G-120

Dictionary
Management

FORTH-83

Format:	FORTH-83
Action:	Verifies that the FORTH-83 standard is supported.

Example: **FORTH-83**   \<cr\> 32-bit Forth 83 ok \<0\>

FORTH
Tool

HERE (- a)

Format:	HERE
Action:	Pushes the address of the next available dictionary location on the stack.

Example: The HERE pointer is incremented by all words which add items to the dictionary :

HEX <cr> ok <$0>

HERE . <cr> 17806 ok <$0>

: DUMMY ; <cr> ok <$0>   (Because the dictionary headers are)
                                  (separated from their corresponding run-time)
HERE . <cr>17808 ok <$0>   (code, the HERE pointer was only incremented)
                                  (by 2 bytes by the definition DUMMY. These two)
                                  (bytes are required to compile the ;)

10 ALLOT <cr> ok <$0>   (ALLOT is used to allocate space in the)
HERE . <cr> 17818 ok <$0>   (dictionary.)

300 W, <cr> ok <$0>   (All of the "comma" words (C, , W, and ,) lay)
HERE . <cr> 17820 ok <$0>   (a value into the dictionary and advance)
                                  (the HERE pointer accordingly.)

See also: ALLOT , , , C, , W,

G-122

Dictionary Management

HEX

Format: HEX

Action: Sets the current task's number base - which affects
                 all number input/output operations - to hexadecimal.
                 A $ sign is included in the stack depth indicator when
                 the current base is hexadecimal.

Example:

DECIMAL <cr> ok <0>

10 <cr> ok <1>

HEX <cr> ok <$1>

. <cr> A ok <$0>

HEX <cr> ok <$0>

: NUMBERS CR 10 0 DO I . LOOP ; <cr> ok <$0>

NUMBERS <cr>

0 1 2 3 4 5 6 7 8 9 A B C D E F ok <$0>

The definition of HEX is -

DECIMAL <cr> ok <0>

: HEX 16 BASE ! ; <cr> ok <0>

See also: DECIMAL , BASE , BINARY

G-123

Number
I/O

HOLD (c -)

Format:	<# . . . c HOLD . . . #>
Action:	Inserts the character whose ASCII value is on top of the stack into the next available position in the formatted ASCII string being constructed. HOLD must be used within <# and #>.

Example: In the example used in the # glossary entry, HOLD was used to insert a decimal point and a dollar sign into the formatted strings produced by $String :

`: $String ( n - ) <# # # ASCII . HOLD #S ASCII $ HOLD #> TYPE ; <cr> ok <0>`

`74638 $String  <cr> $746.38 ok <0>`

DATE uses HOLD to insert the backslash character into a formatted string:

`: DATE ( n - ) <# # # ASCII / HOLD # # ASCII / HOLD # # #> TYPE ; <cr> ok <0>`

`100961  DATE  <cr> 10/09/61 ok <0>`

A formatted string is always built from right to left (from higher to lower memory locations in the PAD. Each new digit in a formatted string is placed in the next lower byte location in the PAD:

<---PAD				PAD + 84 bytes--->
		/	6	1

This diagram shows the formatted string under construction. The backslash has just been inserted by HOLD into the date string.

See also: <# , # , #S , SIGN , #>

G-124

Number I/O

I_____(- n)_____

Format:	I
Action:	When inside of a DO. . . LOOP or DO. . . +LOOP, I puts the current loop index value on the parameter stack. A more general description is that I is moving a copy of the top of the loop stack to the parameter stack. However, the word R@ (which also moves a copy of the top of the loop stack to the parameter stack) is usually used to perform this function when outside of DO loops.

Example:

    : NUMBERS 5 0 DO I . LOOP ;

                  1   2   3 4

    NUMBERS   <cr> 0 1 2 3 4 ok

                1   2   3   4

1. Parameter and loop stack right before DO:

2. Parameter and loop stack right after DO:

3. Paramater and loop stack after 4 times through loop:

4. Parameter and loop stack after loop completed:

See also: R@ , >R , R> , I'

G-125

Program Control Structure

I' (- n)

" i-tick "

Format: I'

Action: In the general case, I' puts a copy of the second item on the loop stack on the parameter stack. When used inside of a loop, I' places the limit value (which will be the second item on the stack when a DO loop is executing) for the loop on the parameter stack.

Example: This example will generate and print one row of a multiplication table :

```
: *TABLE                    (The index of the loop
   CR                        is multiplied by the
   11  1  DO                 limit of the loop and
      I I' * CR   ◄───────    the result printed on
   LOOP ;  <cr> ok <0>        the screen each time
                              through the loop. )
```

```
*TABLE   <cr> 11 22 33 44 55 66 77 88 99 110 ok <0>
```

The loop stack at this point would be:

(Loop index)   **6**   ◄─── (I gets this value.)

(Loop limit)   **1 1**   ◄─── (I' gets this value.)

Loop Stack

See Also: DO , LOOP , +LOOP , I , I'

G-126

Program Control Structure

IF   (f -)

Format:	IF may be used in two different formats:

```
    flag
    IF
       ( code executed if flag passed to IF is true (non-zero) )
    THEN
     ( code which is always executed )

or  flag
    IF
       ( code executed if flag passed to IF is true (non-zero) )
    ELSE
       ( code executed if flag passed to IF is false (zero) )
    THEN

       ( code which is always executed )
```

Action: IF is the first part of the IF...THEN or the IF...ELSE...THEN program control structures. IF expects on the stack a flag which indicates whether or not the code immediately following the IF should be executed.

Example:

```
2  CONSTANT  VERY  <cr> ok  <0>

: FAREWELL  ( n - )
      ." Have a "
      VERY  >
      IF
             ." very "
      THEN
      ." good day !"    ;  <cr> ok  <0>
```

1 FAREWELL   <cr> Have a good day ! ok <0>

3 FAREWELL   <cr> Have a very good day ! ok <0>

See Also: ELSE , THEN

G-127

Program
Control
Structure

IL   (a n -)

Format: start-address #instructions-to-disassemble IL

Action: Symbolically disassembles n instructions starting
at address a.

Example: IL can be used to disassemble user-defined words
and MACH2 kernel words. Note in the disassembly of
the user-defined word 'Test' that user references to
MACH2 kernel words are compiled as 'JSR -xx(A6)'
instructions. The '-xx' is the offset from the address
in the A6 register to the jump table entry for the kernel
word being referenced. A user reference to a kernel
word will always be compiled as a 'JSR' through the jump
table. Note also that '."' or '"' string data are displayed
properly:

```
:  Test  (  - n )   3  4500  " ."  string"  ;  <cr> ok <0>
'  Test  8  IL  <cr>
055230:    MOVEQ.L    #$3,D0
055232:    MOVE.L     D0,-(A5)
055234:    MOVE.L     #$1194,-(A5)
05523A:    JSR        $-7C5E(A6)
05523E:    BSR.S      +$A              ;   $55248
           DC.B       6
           DC.B       'string'
055248:    MOVE.L     (A7)+,-(A5)
05524A:    JSR        $-7FEE(A6)
05524E:    RTS
 ok <0>
```

G-128

**ASSEMBLER
Word**

IMMEDIATE

Format:	: forth-definition . . . . . . ; IMMEDIATE
Action:	Marks the most recently defined dictionary entry as a word which will be executed when encountered during compilation instead of being compiled during compilation. Because immediate words are executed during compilation, they never generate any compiled code.

Example:

: SPEAK-NOW ." Compiling. ." ; IMMEDIATE <cr>

: TEST-STATE
   SPEAK-NOW ." Running. ." ; <cr> Compiling. . ok <0>

   (SPEAK-NOW is an immediate word
   which will be executed whenever
   it is encountered. In this case,
   SPEAK-NOW was encountered during
   the compilation of TEST-STATE and run.
   When SPEAK-NOW is run it displays
   this message on the screeen.)

TEST-STATE <cr> Running. . ok <0>

   (When TEST-STATE is run, only the words which
   generated code during compilation will be executed.
   The compiled code which comprises TEST-STATE has
   no memory of SPEAK-NOW because SPEAK-NOW was
   executed during compilation and thus did not generate
   any compiled code. This is why SPEAK-NOW's message
   was not displayed when TEST-STATE was executed.)

See also: [COMPILE] ,] , [, STATE , SMUDGE

Compilation Word

G-129

J ___ (_ - n) ___

Format:	J
Action:	In the general case, J puts a copy of the third item on the loop stack on the parameter stack. When used inside of a nested loop, J places the index value for the next loop outside of the current loop (which will be the third item on the loop stack while the DO loops are executing) on the parameter stack.

Example: This example creates a few rows of a multiplication table:

```
: *TABLE
  CR
  6 1 DO
        11 1 DO
              I J *              ( Multiplying the
        LOOP .                     index of the inner
  CR LOOP ;  <cr> ok <0>           loop ( I ) by the
                                   index of the outer
                                   loop ( J ).)
```

Outer Loop — Inner Loop

*TABLE <cr>

```
1  2  3  4  5  6  7  8  9  10
2  4  6  8  10 12 14 16 18 20
3  6  9  12 15 18 21 24 27 30
4  8  12 16 20 24 28 32 36 40
5  10 15 20 25 30 35 40 45 50 ok <0>
```

The return stack at this point would be:

(Index for inner loop)	6	⟵ ("I" gets this value.)
(Limit for inner loop)	11	
(Index for outer loop)	4	⟵ ("J" gets this value.)
(Limit for outer loop)	6	

Loop Stack

See Also: DO , LOOP , +LOOP , I , I'

G-131

Program
Control
Structure

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

KEY   (- c)

Format:	KEY
Action:	Waits until a key is pressed and then puts the ASCII value of the character on the stack.

Example:

KEY <cr>_

After KEY is typed in and a carriage return pressed, the cursor still remains on the same line because KEY is running at this point and waiting for a key to be pressed.

If an "M" is pressed KEY will put the ASCII value for an "M" on the stack and complete running.

KEY <cr> ok <0>   (an 'M' was pressed)

DUP <cr> ok <2>

. <cr> 77 ok <1>

EMIT <cr> M ok <0>

See also: EXPECT , EMIT , ?TERMINAL

Character
Input

LAST (- a)

Format: LAST

Action: Pushes the address of the variable which contains the link field address of the most recent dictionary entry on the stack.

G-133

Compilation
Word

LEAVE

Format:	LEAVE
Action:	Exits from within a DO loop. Transfers execution to the word just beyond the next LOOP or +LOOP. The loop is terminated and the loop index and limit are discarded from the loop stack.
	LEAVE may appear within other control structures which are nested within the DO loop structure. More than one LEAVE may appear within a DO loop.

Example: TERMINATE uses LEAVE to terminate the DO loop after the index value has reached 7.

```
:  TERMINATE
   10  0  DO
           I . CR
           I 7 =
           IF
               LEAVE
           THEN
   LOOP ;  <cr> ok  <0>
```

(Using the equal comparison operator to see if the loop index has reached 7.)

(The IF. . .THEN program control structure is nested within the DO. . .LOOP structure.)

```
TERMINATE   <cr>

1
2
3
4
5
6
7 ok  <0>
```

See Also: DO , LOOP , +LOOP , EXIT , IF , THEN

Program
Control
Structure

LINK>BODY   (a1 - a2)

" link-to-body "

Format: link-field-address LINK>BODY

Action: Given a link field address, LINK>BODY will return
the corresponding parameter field address.

See also: BODY>·`K ·

G-135

Compilation
Word

LIST ___(n -)___

Format: block-number LIST

Action: Displays the contents of block n of the current file
on the screen.

Note About Current Files:

In MACH2 there is a system variable called FILEID. Any FORTH Storage I/O words which
interact with open files will look in this location prior to their actions to determine which
file they should be operating on. If a zero is stored in FILEID it means that there is currently
no file to operate upon. Words such as LIST, which determines which file it should
list from by looking in FILEID will return a "No file specified..." error message if used when
a zero is found in FILEID.

If a non-zero value is stored in FILEID it is assumed to be a file reference value which
uniquely identifies which file should be operated on. The file whose file reference number
is stored in FILEID is called the current file. The MACH 2 file interaction words (BLOCK,
BUFFER, LIST, LOAD) will perform their actions on the current file.

After using $OPEN to open a file, you can make the file the current file by storing the file
reference number returned into FILEID:

           DECIMAL   <cr>ok <0>
           4 FileID W! <cr> ok <0>   (Storing a 16-bit file reference number
                                      into FileID. Now the file corresponding
                                      to this file reference number will be the
                                      current file.)

Now LIST could be used to list the contents of a block in this current file.

See also: LOAD

G-136

Storage
I/O

LITERAL (n -)

Format: n LITERAL

Action: During compilation, LITERAL takes the number on top of
                     the stack and compiles it (in FORTH compiled numbers
                     are referred to as literals).

Example: The following example shows a common use of LITERAL :

: SECS/YEAR [365 24 * 60 * 60 *] LITERAL ; <cr>ok <0>

SECS/YEAR is a value to be used often in a program. Instead of having to calculate
the number of seconds in a year each time the program needs the value, SECS/YEAR
calculates the value only once--during compile time--and lays the value in the definition.
This method has two benefits. One is that it reduces execution time. Every time
SECS/YEAR is referenced it pushes the pre-compiled value on the stack, it does not
have to perform any run-time calculations. The second benefit is increased program
readability. The above definition of SECS/YEAR is much more readable than the
following definition which simply includes the precalculated value.

: SECS/YEAR 31536000 ; <cr> ok <0>

See also:] , [

G-137

Compilation
Word

LOAD (n -)

Format:	block-number LOAD
Action:	Uses BLOCK to copy the specified block from storage to a block buffer in memory if it is not already in memory. LOAD then redirects the interpreter so that it interprets words in the block buffer rather than words in the input message buffer. When all of the words in the block buffer have been interpreted (LOADed), the interpreter is switched back to interpreting words from the input message buffer. Operates on the currently open file.

Since source files in MACH 2 are arranged as text files rather than in block-size screens, the word LOAD is not usually used to load files. The word INCLUDE" would be used to load a file from within a program. |

See also: LIST

Storage
I/O

LOOP

Format:

limit index
DO
(code to be executed each time through loop)
LOOP

Action:

LOOP is the smart half of the DO. . .LOOP program control
structure. LOOP expects to find a limit and an index on
the loop stack. The number of times the loop will be
executed is determined by the limit and index using
the formula: limit - index = # of loops. Each time LOOP
is executed it increments the loop index by one and compares
the new index value to the limit value. If the index is
less than the limit the loop will be executed again. If
the index is greater than or equal to the limit, LOOP will
terminate the loop by removing the limit and index
from the loop stack and allowing program execution to
continue on to the code following the LOOP.

Example:

: NUMBERS 5 0 DO I . LOOP ; <cr> ok <0>

NUMBERS   <cr> 0 1 2 3 4 ok <0>

The effect of LOOP -

Loop stack before first LOOP:
Index 0
Limit 5
Loop Stack

Loop stack after first LOOP:
1 Index
5 Limit
Loop Stack

Loop stack after last LOOP:
Loop Stack

See also: DO , LOOP , +LOOP , I , I' , J

G-139

Program Control Structure

L_EXT   (n1 - n2)

" long-extend"

Format: 16-bit-value L_EXT

Action: Extends the word-length (16-bit) value in the lower two bytes
                of the number on top of the stack into a long-word (32-bit)
                value by copying bit 15, the sign bit for a word-length value,
                to bits 16-31 of the long-word value.

Example: HEX   <cr> ok <$0>

                FF89   L_EXT .   <cr> -77 ok <$0>

                89   L_EXT .   <cr> 89 ok <$0>

G-140

Number
I/O

MACH

Format: MACH

Action: Used to toggle the MACH bit on the last word defined (its use
                 is similar to IMMEDIATE or SMUDGE). If a word with its MACH
                 bit set is encountered during compilation all of the assembly
                 language instructions which comprise the word will be laid
                 into the definition being compiled. Normally, a jump to the
                 executable code for a word (JSR) would be compiled. This
                 can be used to increase speed and, at times, save space.

                 Look in the appendices for a list of precautions to observe
                 when using MACH.

See also: IMMEDIATE , SMUDGE

G-141

Compilation
Word

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

MACHMODULE

Format: MACHMODULE <main-word> <module name>

Action: MACHMODULE creates user trap modules to be
           shared by MACH2 programs and applications.
           MACHMODULE should be used after the code to be
           placed in the module has been located into memory.
           <main word> is the word which will be run when the trap
           module is later accessed. <module name> is the name
           for the module.

           When MACHMODULE is executed MACH2 takes all
           code in the user's code area and writes it out to a new
           trap module with the given name. After MACHMODULE
           has completed, MACH2 will return to the OS-9 shell.
           At this point you may perform a directory listing to see
           that the new trap module does exist.

See also: ASSIGNMODULE , TCALL

OS-9
Interface

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

MAKEMODULE

Format: MAKEMODULE <main-word> <module name>

Action: MAKEMODULE creates user trap modules to be shared by non-MACH2 programs and applications. MAKEMODULE should be used after the code to be placed in the module has been located into memory. <main word> is the word which will be run when the trap module is later accessed. <module name> is the name for the module.

When MAKEMODULE is executed MACH2 takes all code in the user's code area and writes it out to a new trap module with the given name. After MAKEMODULE has completed, MACH2 will return to the OS-9 shell. At this point you may perform a directory listing to see that the new trap module does exist.

See also: ASSIGNMODULE , TCALL

G-143

OS-9
Interface

MATH

Format:	ONLY MATH or ALSO MATH
Action:	MATH is the vocabulary which contains all of the floating point math words.

When used with ONLY, MATH will become the only vocabulary included in the current search order .

When used with ALSO, the MATH vocabulary will be appended to the current search order. This will cause MATH to become the transient vocabulary (the vocabulary which is searched first). Any other vocabularies in the search order will be searched after the transient vocabulary. WORDS will only display the words in the transient vocabulary and the words used to specify the search order when executed.

Example:

To specify a search order in which the FORTH vocabulary will be searched first and the MATH vocabulary second :

ONLY MATH   **<cr>** ok <0>
ALSO FORTH   **<cr>** ok <0>

The word ORDER will display the current search order and the vocabulary to which new definitions are being appended :

ORDER   **<cr>**
Search Order   : FORTH   MATH
Definitions   : FORTH
 ok <0>

To make the FORTH vocabulary the only vocabulary which is searched :

ONLY MATH   **<cr>** ok <0>

See also: ONLY , ALSO , ORDER , DEFINITIONS

Dictionary
Management

MAX   (n1 n2 - n3)

Format: n1 n2 MAX

Action: Compares the two numbers on top of the stack and
leaves the greater.

Example:

5 10 MAX <cr> ok <1>   5 -5 MAX <cr> ok <0>

. <cr> 10 ok <0>   . <cr> 5 ok <0>

For Assembly Language Programmers:

```
CODE  MAX ( n1  n2 - n3 )
      MOVE.L    (A5)+,D0
      CMP.L     (A5),D0
      BLE.S     @1
      MOVE.L    D0,(A5)
@1    RTS
END-CODE
```

See also: MIN

G-145

**Comparison
Operator**

MIN   (n1 n2 - n3)

Format: n1 n2 MIN

Action: Compares the two numbers on top of the stack and
                     leaves the lesser.

Example:

38 42 MIN   \<cr\> ok \<1\>   38 37 MIN   \<cr\> ok \<1\>

. \<cr\> 38 ok \<0\>   . \<cr\> 37 ok \<0\>

For Assembly Language Programmers:

```
CODE  MIN  ( n1  n2  -  n3 )
        MOVE.L      (A5)+,D0
        CMP.L       (A5),D0
        BGE.S       @1
        MOVE.L      D0,(A5)
@1    RTS
END-CODE
```

See also: MAX

G-146

Comparison
Operator

MOD (n1 n2 - n3)

Format: n1 n2 MOD

Action: Divides n1/n2, leaving the 32-bit remainder from the division on the stack.

Example: **25 7 MOD** <cr> ok <1>

             . <cr> 4 ok <0>

See also: /MOD , */MOD

G-147

Arithmetic
Operator

NEGATE   (n - -n)

Format:	n NEGATE
Action:	Changes the sign of the number on top of the stack (2's complement). The negation of n = 0 - n.

Example: 3 NEGATE <cr> ok <3>

. <cr> -3 ok <0>

For Assembly Language Programmers:

```
    CODE  NEGATE  ( n - -n )
          NEG.L   (A5)
          RTS
    END-CODE
    MACH
```

See also: ABS , DNEGATE

Arithmetic
Operator

NOT   (n1 - n2)

Format: n NOT

Action: Takes the one's complement of the number on top of the stack.

Example: BINARY <cr> ok <%0>
                     10111101   NOT   <cr> ok <%0>
                     . <cr> 1000010 ok <%0>

For Assembly Language Programmers

```
CODE NOT   ( n1 - n2 )

    NOT.L      (A5)
    RTS

END-CODE
MACH
```

See also: AND , OR , XOR

G-149

Logic
Operator

NP (- a)

" name-pointer "

Format: NP

Action: Variable which holds the pointer to the next
               available spot in the names space.

See Also: HERE , VP

G-150

**Dictionary
Management**

NUMBER? (a - n f)

Format:	address-of-string-to-be-converted NUMBER?
Action:	Tries to convert an ASCII string to a binary number with regard to the current number base. NUMBER? expects on the stack the address of the ASCII string to be converted. The first byte of the string, normally the length byte, is skipped over by NUMBER? but must be included. The last byte in the string must be a null or a space. If the conversion is successful and either valid punctuation (commas, periods, back slashes, or semi-colons) or no punctuation is found in the string, a 32-bit value will be put on the stack. If the conversion is unsuccessful, usually due to invalid punctuation or characters within the string, an error message is issued.

The string may contain a preceding minus sign.

Example: In the following example, EXPECT is used to put a number string into a storage area. NUMBER? takes the address of this storage area and converts it contents to a numeric value:

VARIABLE NumString 20 VALLOT <cr> ok <0>
NumString 1+ 19 EXPECT <cr> 123,45<cr> ok <0>

NumString NUMBER? <cr> ok <2>
. . <cr> -1 12345 ok <0>

Notice that the first byte of the storage area was not used since NUMBER? will ignore it. Also, a comma was included in the string entered but NUMBER? did not complain since a comma is a valid punctuation mark.

NOTE: MACH 2 supports 32-bit integers. Punctuation does not generate 64-bit double length numbers. All numbers are 32-bit.

See also: CONVERT , BASE

G-151

Number
I/O

OF   (n1 n2 - (n1))

Format: OF is used in the following format:

                   CASE
                   n   OF (code executed if n is matched) ENDOF
                   n1 OF (code executed if n1 is matched) ENDOF
                   n2 OF (code executed if n2 is matched) ENDOF
                                        .
                                        .
                   nn OF (code executed if nn is matched) ENDOF
                   (code executed if no match was made above)
                   ENDCASE

Action: OF marks the beginning the OF...ENDOF structure
                   used within a CASE statement.

See also: CASE , ENDOF , ENDCASE

**Program
Control
Structure**

ONLY

Format:	ONLY vocabulary-name
Action:	Makes the vocabulary specified by name the only vocabulary searched in all subsequent dictionary searches. Only the words in the specified vocabulary and the words used to specify and change the search order will be found after the use of ONLY. ALSO may be used to append other vocabularies to the search order.

Example:

```
ONLY  OS-9  <cr>  ok  <0>
WORDS  <cr>
$CREATE
$DELETE
$CLOSE
$OPEN
$WRITE
FILEID

WORDS
ORDER
ALSO
ONLY
SEAL
DEFINITIONS
IL
MATH
OS-9
ASSEMBLER
FORTH
 ok  <0>
```

After ONLY OS-9 these are the only words which will be able to be found in any subsequent dictionary search.

3 DUP <cr> DUP ?	(The FORTH vocabulary is not being searched so the system does not know about DUP.)
ALSO FORTH <cr> ok <0>	(Now the FORTH vocabulary will also be searched.)
3 DUP <cr> ok <2>	(And now the system is able to find DUP in the dictionary.)

See also: ALSO , ORDER

G-153

Dictionary Management

OR   (n1 n2 - n3)

Format: n1 n2 OR

Action: Performs the bit-by-bit logical OR of n1 with n2. Leaves
                     the result on top of the stack.

Example: The truth table for OR is:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

BINARY   <cr> ok <%0>

10101010   <cr> ok <%1>

00001111   OR   <cr>ok <%1>

. <cr> 10101111 ok <%0>

DECIMAL   <cr> ok <0>

For Assembly Language Programmers:

```
CODE  OR  ( n1  n2  -  n3 )
      MOVE.L     (A5)+,D0
      OR.L       D0,(A5)
      RTS
END-CODE
MACH
```

See also: XOR , AND , NOT , BINARY , DECIMAL

G-154

Logic
Operators

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

ORDER

Format: ORDER

Action: Search order utility which displays the names of the
                 vocabularies in the current search order and the name
                 of the vocabulary to which new definitions are currently
                 being appended.

Example: This is the initial search order :

ORDER **<cr>**
Search Order : FORTH
Definitions   : FORTH
ok <0>

See also: ONLY , ALSO , DEFINITIONS , WORDS

G-155

FORTH
Tool

OS-9

Format:	ONLY OS-9 or ALSO OS-9
Action:	OS-9 is the vocabulary which contains all OS-9 specific words.

When used with ONLY, OS-9 will become the only vocabulary included in the current search order .

When used with ALSO, the OS-9 vocabulary will be appended to the current search order. This will cause OS-9 to become the transient vocabulary (the vocabulary which is searched first). Any other vocabularies in the search order will be searched after the transient vocabulary. WORDS will only display the words in the transient vocabulary and the words used to specify the search order when executed.

Example:

To specify a search order in which the OS-9 vocabulary will be searched first and the FORTH vocabulary second :

ONLY FORTH <cr> ok <0>
ALSO OS-9 <cr> ok <0>

The word ORDER will display the current search order and the vocabulary to which new definitions are being appended :

ORDER <cr>
Search Order : OS-9 FORTH
Definitions   : FORTH
 ok <0>

To make the OS-9 vocabulary the only vocabulary which is searched :

ONLY OS-9 <cr> ok <0>

See also: ONLY , ALSO , ORDER , DEFINITIONS

G-156

Dictionary
Management

OVER   (n1 n2 - n1 n2 n1)

Format: n1 n2 OVER

Action: Puts a copy of the second item on the parameter stack
                     on top of the stack.

Example: 5 6 OVER <cr> ok <3>

                     .S <cr> 5 6 5 <- TOP ok <3>

For Assembly Language Programmers:

        CODE OVER (n1 n2 - n1 n2 n1)
            MOVE.L   4(A5),-(A5)
            RTS
        END-CODE
        MACH

See also: DUP , DROP , PICK , ?DUP

Stack
Manipulation

PAD (- a)

Format: PAD

Action: Puts on the stack the address of the first byte of the PAD
                   scratch area in memory. The PAD should be treated strictly
                   as a temporary scratch area. It should not be used for storage
                   since many FORTH words use the PAD while performing their
                   operations.

Example: The PAD may be used to hold temporary input strings :

PAD 10 EXPECT <cr> ABCDEFGHIJ ok <0>

PAD 10 TYPE <cr> ABCDEFGHIJ ok <0>

Keep in mind that the number formatting words (which are used for all numeric
output commands) use the upper portion of the PAD.

See also: <# , EMIT

**System/Local
Variable**

PICK   (n - n)

Format:	n PICK
Action:	Moves a copy of the nth item down on the stack (where n = 0 refers to the item on top of the stack) to the top of the stack.

Example:

10 21 32 43 54 65 76 <cr> ok <7>

.S <cr> 10 21 32 43 54 65 76 <- TOP ok <7>

      6th item   2nd item   0th item

2 PICK <cr> ok <8>

.S <cr> 10 21 32 43 54 65 76 54 <- TOP ok <8>

0 PICK <cr> ok <9>

.S <cr> 10 21 32 43 54 65 76 54 54 <- TOP ok <9>

(Note that 0 PICK is equivalent to a DUP.)

Stack before PICK -	Stack after 2 PICK -	Stack after 0 PICK -

See also: DUP , DROP , OVER , ROLL

G-159

Stack Manipulation

QUERY

Format: QUERY

Action: Transfers characters from the keyboard into the terminal
                         input buffer until either 72 characters or a carriage return
                         is received. QUERY sets the >IN system variable to zero and
                         the sets the #TIB system variable equal to the number of
                         characters received.

Example: The definition of QUERY is :

: QUERY

   SPAN @ >R   Since QUERY calls expect it will corrupt the current
                                         value of SPAN. Here the current value of SPAN is
                                         being stored temporarily on the loop stack.

   TIB 72 EXPECT   EXPECT puts up to 72 characters in the TIB.

   0 >IN !   A zero value in the >IN system variable indicates that
                                         the current input stream is the TIB.

   SPAN @ #TIB !   The system variable SPAN will contain the actual
                                         number of characters received by EXPECT. This
                                         number is stored in the system variable #TIB. WORD
                                         uses the contents of #TIB to determine when it has
                                         finished processing words in theTIB.

   R> SPAN ! ;   Restoring the previous SPAN value.

See also: EXPECT , KEY , #TIB , >IN , TIB , QUIT

Character
I/O

QUIT

Format: QUIT

Action: QUIT is the word which resets the system (except for the
                           parameter stack) and starts FORTH running again. When
                           QUIT is executed, it clears the return stack and puts the
                           system in the interpreting state. The system at this point
                           is waiting for keyboard input. The "ok" message will not
                           be seen until a word (or a series of words) have been
                           successfully executed.

Example: The basic definition of QUIT is as follows:

```
: QUIT
  BEGIN
    ( clear return stack )
    QUERY
    ( INTERPRET the input stream )
    ." ok" CR
  AGAIN
; <cr> ok <0>
```

See also: QUERY

Program
Control
Structure

G-161

R>   (- n)

"r-from"

Format:	R>
Action:	Moves the number on top of the loop stack to the top of the parameter stack.

Example:

5 >R <cr> ok <0>

R> . <cr> 5 ok <0>

Paramater stack and loop
stack before R> -

Parameter stack and loop
stack after R> -

Parameter Stack	Loop Stack	Parameter Stack	Loop Stack

For Assembly Language Programmers:

```
CODE R> ( - n )
        MOVE.L   D6,-(A5)
        MOVE.L   D5,D6
        MOVE.L   -(A3),D5
        RTS
END-CODE
MACH
```

See also: >R , R@ , I , J

G-162

Stack
Manipulation

R@ ___(- n)___

" r-fetch "

Format: R@

Action: Moves a copy of the top of the loop stack to the top
                 of the parameter stack. R@ and I are equivalent but
                 I is usually used only within DO loops.

Example: 5 >R <cr> ok <1>

                 R@ . <cr> ok <0>

Parameter stack and loop   Parameter stack and loop
stack before R@ -   stack after R@ -

Parameter   Loop   Parameter   Loop
Stack   Stack   Stack   Stack

For Assembly Language Programmers:

```
        CODE  R@  ( - n )
              MOVE.L        D6,-(A5)
              RTS
        END-CODE
        MACH
```

See also: I , J , R> , <R

G-163

**Stack
Manipulation**

RECURSIVE

Format: : <name> RECURSIVE . . . <name> . . . ;

Action: Immediate word which is used within a recursive colon
                           definition to clear the smudge bit of the word being defined
                           so that it may reference itself.

Example: The following example is a recursive evaluation of a
                           Fibonacci number. This example also makes use of
                           local variables.

```
: Fib  RECURSIVE  { n -- }  ( Clears the smudge bit of Fib so that Fib may )
          n 2 <                ( reference Fib recursively and )
          IF                   ( specifies that 1 named input parameter )
               1               ( - initialized local variable - is set up for this )
          ELSE                 ( definition.)
               n 1- Fib
               n 2- Fib
               +
          THEN  ;           ( ; normally clears the smudge bit for a completed
                              definition.  In this case RECURSIVE has already
                              made the word visible so ; has no additional effect.)
```

See also: : , ; , SMUDGE

Compilation
Word

G-164

REPEAT

Format:

```
BEGIN
  ( code to be executed each time through loop )
flag
WHILE
  ( code to be executed while flag is true )
REPEAT
( code executed when loop terminates)
```

Action: REPEAT marks the end of the BEGIN. . .WHILE. . .REPEAT
               program control structure. When a true flag is passed
               to WHILE, code between the WHILE and REPEAT will be
               executed until the REPEAT is encountered. At the REPEAT,
               program execution will be rerouted back to the code
               which follows the BEGIN. If a false flag is passed to
               WHILE the loop will be terminated by allowing program
               execution to continue to the code which follows REPEAT.
               Loop execution continues while the flag is true.

See Also: BEGIN , WHILE , REPEAT , AGAIN

Program
Control
Structure

ROLL   (n -)

Format:	n ROLL
Action:	Rotates the nth item on the parameter stack to the top of the stack (where n = 0 refers to the item on top of the stack.). N must be greater than 0.

0 ROLL is a null.
1 ROLL is the same as a SWAP.
2 ROLL is the same as a ROT.

Example:

10   21   32   43   54   65   76   <cr> ok <7>

.S   <cr> 10 21 32 43 54 65 76 ok <- TOP <7>

      ↑   ↑   ↑

  6th item   2nd item   0th item

3   ROLL   <cr>   ok <7>

.S   <cr> 10 21 32 54 65 76 43 <- TOP ok <7>

6   ROLL   <cr> ok <7>

.S   <cr> 21 32 54 65 76 43 10 <- TOP ok <7>

Stack before ROLL -   Stack after 3 ROLL -   Stack after 6 ROLL -

before	after 3	after 6
76	43	10
65	76	43
54	65	76
43	54	65
32	32	54
21	21	32
10	10	21

See also: PICK , SWAP , ROT

Stack
Manipulation

ROT   (n1 n2 n3 - n2 n3 n1)

"rote"

Format: n1 n2 n3 ROT

Action: Takes the third item on the parameter stack and puts
                 it on top of the stack, shifting the two items that were
                 previously above it downwards.

Example: 3 2 1 ROT <cr> ok <3>

                 .S <cr> 2 1 3 <- TOP ok <3>

See also: SWAP , ROLL

G-167

Stack
Manipulation

SAVE-BUFFERS

Format: SAVE-BUFFERS

Action: Writes all block buffers which are marked as UPDATE'd to
                    disk and then clears the update bit of all the buffers. The
                    buffers remain assigned to their file block. UPDATE and
                    SAVE-BUFFERS should be used frequently when making
                    changes to a file to avoid losing a large amount of work due
                    to an unexpected event such as power loss.

Example: Writing Data to a File

After opening a file and making the file the current file (see the BUFFER glossary page),
data may be written to the file in 1K (1024 byte) chunks.

The word BUFFER will assign one of the four available block buffers to the specified block
within the current file and will return the address of the block buffer in memory. Once you
have the address of this buffer you may write up to 1024 bytes of information into the
buffer in memory. When you have completed putting data into the buffer, you should use
UPDATE to mark the buffer as changed and then use SAVE-BUFFERS to have those
changes written out to the correct block within the current file. For example, to store 1024
bytes of information into the second block of the current file :

```
: Fill1K   ( bufaddr   --   )
         1023  0  DO
                 ASCII  A  bufaddr I +  C!     ( Fill 1024 bytes with the letter A. )
         LOOP
         UPDATE                                ( Mark the buffer as UPDATEd. )
          SAVE-BUFFERS      ;                  ( Save the changes to the file on disk. )

1 BUFFER  Fill1K  <cr>                          ( Ask for a buffer to be assigned to block
                                                  1-the 2nd block-of the file. Pass the
                                                  address of the buffer to Fill1K. )
```

See also: FLUSH , EMPTY-BUFFERS , UPDATE , BUFFER

G-168

Storage
I/O

SEAL

Format: SEAL

Action: Freezes the current search order and removes
             the link to the words used to change the search
             order.

G-169

**Dictionary
Management**

SIGN   (n1 n2 - n2)

Format: <# . . . . SIGN #>

Action: Uses the sign bit of the 2nd number on the stack to determine
                           if a minus sign should be inserted in the next available position
                           in the formatted ASCII string being constructed.

Example: The word . , which prints out the signed value of the number on
                           top of the stack, uses SIGN :

    : . (n -)

       DUP ABS   (Duplicate the number and take the absolute value of the copy.
                                       A positive value should always be passed to the number
                                       formatting operators.)

       <# #S   (Start the number conversion process. Convert all digits
                                       in the number to ASCII and insert them in the string.)

       SIGN   (Get the second number on the stack. If it is negative, insert
                                       a minus sign at this point.)

       #> TYPE   (Finish the number conversion process and TYPE out the
                                       string.)

       SPACE   ;

The definition of SIGN is:

: SIGN (n1 n2 - n2)

       SWAP   (Put the second value on top of the stack. The 2nd
                                       number is the number used to specify the sign.)

       0< IF   (Is the number negative ?)

       ASCII - HOLD   (If it is, put the ASCII value for a '-' on the stack
                                       and insert it into the string using HOLD.)

       THEN   ;   (If it isn't negative, just exit.)

See also: <# , # , #S , HOLD , #>

G-170

Number
I/O

SMUDGE

Format: SMUDGE

Action: Toggles the smudge bit of the most recently defined dictionary
                     entry. When the smudge bit is set, the definition is invisible
                     to any dictionary search.   : (COLON) sets the smudge bit to
                     make the dictionary entry being built invisible until it has been
                     completed. This ensures that no half-built definitions are found
                     and executed. RECURSIVE clears the smudge bit of the
                     recursive definition being built so that the definition may
                     reference itself.

Example:

    : ADD-FIVE (n -) 5 + ; SMUDGE <cr> ok <0>

                                                        SMUDGE toggles the current
    : sets the smudge   ; clears the smudge   state of the smudge bit. Since
    bit so that ADD-FIVE   bit when the definition   ; has just cleared the smudge
    cannot be found   is completed. Now   bit, this SMUDGE sets it again
    until it is completed. ADD-FIVE may be   so now ADD-FIVE may not be
                           found.   found.

    ADD-FIVE <cr> ADD-FIVE ?

 (Since ADD-FIVE was made invisible by SMUDGE,
   it could not be found in the dictionary search.)

   The definition of RECURSIVE uses SMUDGE :
   : RECURSIVE SMUDGE ; IMMEDIATE <cr> ok <0>

   See also: : , ; , RECURSIVE , IMMEDIATE

Compilation Word

G-171

SPACE

Format: SPACE

Action: Print one space.

Example: The following word will print the numbers 1 through 10 out on a single line -

: NUMBERS   CR 11   1   DO   I   .   LOOP   ; ok <0>

NUMBERS <cr>
1 2 3 4 5 6 7 8 9 10 ok <0>

Compare the previous output to the output of the next word which uses SPACE to insert two extra spaces between each number -

: NUMBERS2
     11 1 DO I . SPACE SPACE LOOP ;   <cr> ok <0>

NUMBERS2 <cr>

1 2 3 4 5 6 7 8 9 10 ok <0>

See also: SPACES , EMIT

G-172

Character
I/O

SPACES (n -)

Format: number-of-spaces-desired SPACES
Action: Prints n spaces.

Example: A high-level definition of SPACES is :

        : SPACES (n -) 0 DO SPACE LOOP ; <cr> ok <0>

     : STAIRCASE 10 0 DO I SPACES I . CR LOOP ; <cr> ok <0>

     STAIRCASE <cr>
     0
      1
       2
        3
         4
          5
           6
            7
             8
               9 ok <0>

NOTE: Because SPACES uses the DO...LOOP structure it will always produce at
least one space (even if passed a zero).

See also: SPACE , EMIT

                         G-173

Character
I/O

SPAN (- a)

Format: SPAN

Action: Puts the address of the system variable containing the
                 number of characters actually received during the last
                 execution of EXPECT.

Example: **VARIABLE CHARS 10 VALLOT** <cr> ok <0>

                 CHARS 10 EXPECT <cr> **ABCDEF<cr>** ok <0>

                 SPAN @ . <cr> 6 ok <0>

See also: EXPECT

System/Local
Variable

SQRT (n1 - n2)

Format: n1   SQRT

Action: Takes the square root of the number on top of the stack
                     and returns the integer result.

Example: **25   SQRT** . <cr> 5 ok <0>
                     30   SQRT . <cr> 5 ok <0>

SQRT will return a zero if it is passed a negative number.

See also: *

**Arithmetic
Operator**

STATE (- a)

Format: STATE

Action: Puts the address of a variable which indicates the
                 state of the system (either compiling or interpreting)
                 on the stack. A non-zero value in the STATE variable
                 indicates compilation is occuring. A zero value in
                 STATE indicates that all input is being interpreted
                 and executed immediately.

Example: : STATE@ STATE @ . ; <cr> ok <0>

                 IMMEDIATE   <cr>ok <0>

                 STATE@   <cr> 0 ok <0>

                 (Because the system is executing
                   words immediately, the STATE
                   variable contains a zero.)

: CHECK-STATE   STATE@   ; <cr> -1 ok <0>

                 (STATE@ is an immediate word, so
                   the contents of the STATE variable are
                   being examined during the compilation
                   of the word CHECK-STATE. The contents
                   of STATE at this time are non-zero,
                   indicating a compilation state.)

See also:] , [

G-176

Compilation
Word

SWAP   (n1 n2 - n2 n1)

Format: n1 n2 SWAP

Action: Exchanges the top two items on the stack.

Example: 2 1 SWAP   \<cr> ok \<2>

              .S   \<cr> 1 2 \<- TOP ok \<2>

For Assembly Language Programmers:

```
CODE  SWAP  ( n1  n2  -  n2  n1 )
      MOVE.L    (A5)+,D0
      MOVE.L    (A5),-(A5)
      MOVE.L    D0,4(A5)
      RTS
END-CODE
MACH
```

See also: ROT , ROLL

G-177

Stack
Manipulation

TCALL

Format: TCALL <vector#>,<function code>

Action: Assembler word used to generate user trap calls. User trap calls
are used to access user trap handler modules and the OS-9 math
and CIO library modules.

TCALL should be followed by the trap vector number used to
access the module and the function code to be passed to the
module.

Trap vector #0 is reserved by OS-9 for calls to the kernel module.
Trap vectors #14 and #15 are used to access the OS-9 math
modules. MACH2 uses trap vector #13 to access the MACH2
disassembler/debugger trap handler module.

Example: A very simple trap handler module is created below to demonstrate
the use of TCALL. The module will add 2 to the function code
passed to it and return the result to the calling program.
MACHMODULE will return to the shell after it has finished creating
this new trap module:

```
: Simple ( functioncode - functioncode+2 )  2+  ;
MACHMODULE  Simple  SimpleModule
```

The following commands should be typed after you
have re-entered MACH:

```
5  CONSTANT  SimpleVector
" SimpleModule" 1+    SimpleVector  ASSIGNMODULE

: Alpha ( - n )    TCALL SimpleVector,1 ;
: Beta  ( - n )    TCALL SimpleVector,2 ;
: Gamma ( - n )    TCALL SimpleVector,8 ;

Alpha . <cr> 3 <ok>
Beta . <cr> 4 <ok>
Gamma . <cr> 10 <ok>
```

See also: OS9

OS-9
Interface

THEN

Format: THEN may be used in two formats:

>   IF
>   (code executed if a true flag is passed to IF)
>   THEN
>
> or   IF
>   (code executed if a true flag is passed to IF)
>   ELSE
>   (code executed if a false flag is passed to IF)
>   THEN

Action: THEN marks the end of the IF...ELSE...THEN or IF...THEN
                 program control structures. Program execution will
                 always be routed to the code following the THEN,
                 regardless of what flag is passed to the IF.

See Also: IF , ELSE

**Program
Control
Structure**

! ˚ # $ % & ' ( ) ˚ + , · . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

TIB   (- a)

"t-i-b"

Format: TIB

Action: Puts the address of the text input buffer on the stack. The
                 text input buffer receives all keyboard input. BLK, >IN,
                 #TIB and TIB are the four system variables responsible for
                 maintaining control of the input stream.

                 QUERY always puts incoming characters into the TIB.

See also: BLK , >IN , #TIB

G-180

System/Local Variable

TIMER

Format: COUNTER... <words to be timed>...TIMER

Action: Takes the contents of the variable initialized by
                   COUNTER and displays the execution time in
                   seconds and ticks. The time display is always
                   in DECIMAL. The previous base is restored
                   after TIMER is finished.

See also: COUNTER

G-181

FORTH
Tool

TURNKEY

Format:	TURNKEY <main-word> <application name>
Action:	Creates a complete stand-alone application with the specified name which will run the specified FORTH word upon start-up. The application may be run from the OS-9 shell.
	The minimum application size is 6K. The application is given a $4000 byte stack area (shared by all of the stacks) and a $4000 byte variable area.

<u>Example:</u>

```
: <main-word> ( - )

    CR
    ." Press 'R' for RED"

    KEY ASCII R =
    IF
        ." Red"
    ELSE
        ." Blue"
    THEN
    BYE ;

    TURNKEY <main-word> Colors <cr>
```

After MACH2 returns to the shell you may type 'COLORS'
at the shell prompt to run the application.

The dictionary headers are not included in a turnkeyed application so the application must not reference any FORTH words which require the headers presence. Also, once an application has undergone the TURNKEY process, the dictionary is "frozen". No words which alter the dictionary (W, ALLOT and any other words in the compiler segment) in any way may be used by the application program.

To make sure you are not using any compiler words in your application program store a non-zero value in the system variable VERBOSE and then load your program. While VERBOSE is 'on' warning messages will be issued whenever a word in the compiler segment is referenced.

See also: MAKEMODULE , MACHMODULE

Application
Word

G-182

TYPE   (a n -)

Format:	address-of-characters #-characters-to-type TYPE
Action:	Prints the n characters stored starting at the specified address out to the current output device.

Example: COUNT takes the address of a string on the stack and returns the length of the string and the address of the actual first byte in the string on the stack for TYPE :

" Hello there !" COUNT TYPE <cr> Hello there ! ok <0>

The number formatting word #> also sets up the stack for TYPE :

```
: PHONE# ( n - )
        <# # # # # ASCII - HOLD # # # #> TYPE ; <cr> ok <0>
```

4445678   PHONE#   <cr> 444-5678 ok <0>

See also: COUNT , #> , PAD

G-183

Character
I/O

U. (u -)

" u-dot "

Format: u U.

Action: Prints the unsigned single-length number on top of the
                   stack on the screen followed by one space.

Example: U. pays no attention to the sign bit of the number on the stack :

                   HEX   <cr> ok <$0>

                   -2   U.   <cr> FFFFFFFE ok <$0>

This is the definition of U. :

: U. (n -)   <# #S #> TYPE SPACE ; <cr> ok <0>

See also: . , SIGN , #S

G-184

Number
I/O

U<   (u1 u2 - f)

" u-less-than "

Format: u1 u2 U<

Action: Compares the two unsigned numbers on top of the stack.
                     Returns a true flag if u1 is less than u2, and a false flag
                     if u1 is not less than u2.

Example: This example demonstrates the difference between using
                     the signed < and the unsigned U<.

                     HEX   <cr> ok <$0>

                     -1 A000 . . <cr> A000 -1 ok <$0>

                     -1 A000 < . <cr> -1 ok <$0>

                     -1 A000 U. U. <cr> A000 FFFFFFFF ok <$0>

                     -1 A000 U< <cr> 0 ok <$0>

See also: < , U. , D<

G-185

**Comparison
Operator**

UM* (u1 u2 - u3)

" u-m-times "

Format: u1 u2 UM*

Action: Multiplies two unsigned numbers and leaves the
                           unsigned result on top of the stack.

Example: HEX   <cr> ok <$0>

                          FFFFFFFF 2 *   <cr> ok <$1>

                          . <cr> -2 ok <$0>

                          FFFFFFFF 2 UM* <cr> ok <$1>
                          .S <cr> 1 <- TOP ok <$1>

See also: U< , U. , *

Arithmetic
Operator

UM/MOD   (u1 u2 - u3 u4)

" u-m-slash-mod"

Format: u1 u2 UM/MOD

Action: Divides u1 by u2 and leaves the unsigned quotient on top
                  of the stack and the unsigned remainder below it.

Example: In the first example signed division is performed. FFFFFFFF is a
                  -1 in signed arithmetic so -1 divided by 5 leaves a quotient of 0 and
                  a remainder of -1. The second example uses an unsigned division.
                  With unsigned division the FFFFFFFF is not treated as a -1 and the
                  results are a quotient of 33333333 and a remainder of 0.

HEX   <cr> ok <$0>

FFFFFFFF 5 /MOD <cr> ok <$2>

.S <cr> -1 0 <- TOP ok <$20>

FFFFFFFF 5 UM/MOD <cr> ok <$2>

.S <cr> 0 33333333 <- TOP ok <cr>

See also: UM*, /MOD , MOD , */MOD

G-187

Arithmetic
Operator

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

UNTIL   (f -)

Format: BEGIN
                 (code to be executed while flag is false (zero))
                 flag
                 UNTIL

Action: UNTIL is the decision maker in the BEGIN. . .UNTIL
                 program control structure. UNTIL makes its decision
                 on whether or not the loop should be terminated based
                 upon the value of the flag passed to it. If the flag is true
                 (non-zero), UNTIL will terminate the loop by allowing
                 program execution to continue onto the code which
                 immediately follows the UNTIL. If the flag is false (zero),
                 UNTIL will reroute program execution to the code following
                 the BEGIN. Due to the structure of the BEGIN. . .UNTIL
                 loop, the code within the loop will always be executed at
                 least once.

See Also: BEGIN , WHILE , REPEAT , AGAIN

G-188

Program
Control
Structure

UPDATE

Format: UPDATE

Action: Sets the update bit of the most recently accessed block
                   buffer to indicate that the contents have been changed
                   and should be saved.

Example: One way to make a change to a file :

```
"  TestFile.FTH" 1+ 3 $OPEN .  . <cr> 0 4 ok <0>   \ Opening a read/write version of
                                                    \ the file.

4  FILEID  !  <cr> ok <0>                           \ Make the file the current file by
                                                    \ storing its path number in FileID.

2  BLOCK   <cr> ok <1>                              \ This returns the address of the
                                                    \ first character in the 3rd block
                                                    \ of the current file.

100  +  <cr> ok <1>                                 \ Add 100 to the address left by
                                                    \ BLOCK to get the address of
                                                    \ the 100th byte in the block.

33  SWAP  C!  UPDATE <cr> ok <0>                    \ Storing a 33 in the 100th byte of
                                                    \ of the block and mark the most
                                                    \ recently accessed block--
                                                    \ block 2--as UPDATE'd.

SAVE-BUFFERS   <cr> ok <0>                          \ Writing all UPDATE's blocks to
                                                    \ the disk and clearing the
                                                    \ UPDATE bits.
```

UPDATE should be used after modifying a block buffer. Setting the update bit
guarantees that the buffer will be written to disk.

See Also: SAVE-BUFFERS , FLUSH

G-189

Storage
I/O

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

VALLOT _____ (n -) _____

"variable-allot"

Format: #bytes-to-allocate-in-variable-space VALLOT

Action: Allocates additional memory in the variable space for the
most recently defined variable by incrementing the variable
pointer, VP, by the specified number of bytes. Initialization
of storage locations allocated by VALLOT is left up to the
programmer.

It is very important to note the difference between VALLOT
and ALLOT. VALLOT allocates memory in the variable space
while ALLOT allocates memory in the dictionary space.

Example: ?FREE may be used to see the effect on the variable space
of VALLOTing space in the variable space:

```
?FREE  <cr>  ok <0>
Code : 32688          \  Initial status of variable space.
Vars : 13390
Name : 16290
ok <0>


VARIABLE  ARRAY  <cr> ok <0>
?FREE  <cr> ok <0>
Code : 32680          \  VARIABLE automatically allocates
Vars : 13386          \  4 bytes of storage for a new variable.
Name : 16270          \  Notice that the  amount of available
ok <0>                \  variable space has decreased by
                      \  4 bytes from above.


400  VALLOT  <cr> ok <0>
?FREE  <cr> ok <0>
Code : 32680          \  The 400  VALLOT reduced the
Vars : 12986          \  amount of available variable space
Name : 16270          \  by 400 bytes.
ok <0>
```

See also: VP, VARIABLE

G-190

Memory
Operator

VARIABLE

Format: VARIABLE <name>

Action: Defining word which creates a dictionary entry using the
                           specified name and allocates 4 bytes of memory in the
                           variable space for the contents of the variable. The parameter
                           field of the new variable's dictionary entry contains its offset
                           into the variable space using the address in the A6 register
                           as a base. When words created by VARIABLE are executed,
                           the absolute address of their 4-byte storage location is
                           calculated using the address in the A6 register and the offset
                           and is pushed on the stack.

Example: VARIABLE is used in the following manner :

                           VARIABLE   SUM   <cr>ok <0>

                           10   SUM   !   <cr>ok <0>

                           SUM   @   .   <cr>10 ok <0>

                           NOTE: VARIABLE generates IMMEDIATE words that
                           are state dependent. This characteristic allows variables
                           to increase their run-time speed by optimizing their
                           addressing modes at compile time.

See also: VALLOT , VP

                           G-191   Defining
                                                              Word

VERBOSE (- a)

Format:	VERBOSE
Action:	VERBOSE is a tri-state system variable which controls the issuance of certain warning messages during compilation. The chart summarizes the effects of VERBOSE:

VERBOSE Contents Effect

Negative value	Warning messages will be issued whenever a word not allowed in a 'generic' trap module is compiled.
Zero	No action.
Positive value	Warning messages will be issued whenever a word not allowed in a TURNKEY module or a 'MACH' trap module is compiled.

Example:

    1   VERBOSE !   <cr> ok <0>

    : Check QUIT ; <cr>
    QUIT may not be used in a TURNKEY application.

See also: TURNKEY

G-192

FORTH Tool

VERIFY   (- a)

Format: VERIFY

Action: Returns address of system variable whose contents
                 determine whether or not a file will be displayed on
                 the screen during loading. A non-zero value will
                 indicate that the file should be displayed and a zero
                 value will indicate that the file should not be displayed.

FORTH
Tool

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\ ] ^ _ ` a-z { | } ~

VOCABULARY

Format:	VOCABULARY <name>
Action:	Defining word which creates a dictionary entry using the specified name. The dictionary entry marks the base for the new linked-list of words which will be added to this vocabulary. The vocabulary name is used with ONLY and ALSO to modify the search order. Up to 9 vocabularies may be defined by by the user.

Example:

ONLY FORTH <cr> ok <0>   (Make FORTH the only vocabulary
            searched.)

VOCABULARY FONTS <cr> ok <0>   (Create a new vocabulary named
            FONTS .)

ALSO FONTS <cr> ok <0>   (Add the FONTS vocabulary to the
            search order.)

ORDER <cr>   (ORDER shows the current search)
Search Order : FONTS FORTH   (order and the vocabulary to which)
Definitions : FORTH   (new definitions will be appended.)
ok <0>   (FONTS is currently being searched
            first.)

DEFINITIONS <cr> ok <0>   (Definitions will make the vocabulary
            which is currently being searched
            first also the vocabulary to which new
            definitions will be appended.)

ORDER <cr>
Search Order : FONTS FORTH
Definitions : FONTS
ok <0>

See also: DEFINITIONS

G-194

Defining Word

VP   (- a)

" variable-pointer "

Format: VP

Action: Variable which holds the offset that, when added to the
                 contents of the A6 register, points to the next free
                 variable space location.

See Also: VARIABLE , VALLOT

Compilation
Word

W!   (n a -)

" w-store "

Format:	16-bit-value address W!
Action:	Stores the 16-bit value at the specified address.

Example:

The following example shows that even though numbers placed on the stack are represented using 32-bits, the operator W! will take only the least significant word (16-bits) from the stack and store it in memory :

    HEX   <cr> ok <$0>
    VARIABLE STORAGE   <cr> ok <$0>

    132F   STORAGE W!   <cr> ok <$0>

The number 132F as it appears on the stack.

0 0	0 0	1 3	2 F
1 byte	1 byte		

Bytes in memory :

lower memory --> higher memory

STORAGE

W! takes only the lowest-order word from the stack and stores it starting at the specified address.

For Assembly Language Programmers:

```
CODE  W!  ( w  a  -  )
        MOVE.L    (A5)+,A0
        MOVE.L    (A5)+,D1
        MOVE.W    D1,(A0)
        RTS
    END-CODE
    MACH
```

See also: W@

Memory
Operator

W, (w -)

" w-comma "

Format:	n W,
Action:	Lays the low-order word of the 32-bit value on the stack into the dictionary, starting at the address pointed to by the HERE pointer. W, will first check to make sure it is on a word boundary and adjust the HERE pointer if necessary. The HERE pointer is then incremented by 2.

Example:

CREATE TABLE \<cr\> ok \<0\>

10 W, 20 W, 30 W, 40 W, \<cr\> ok \<0\>

TABLE W@ . \<cr\> 10 ok \<0\>

TABLE 2 2* + W@ . \<cr\> 30 ok \<0\>

See also: , , C, , ALLOT , HERE

G-197

Compilation
Word

W @ _ (a - w)

" w-fetch "

Format: address W@

Action: Replaces the address on top of the stack by the word-length
16-bit) value which is stored at that address. The upper
2 bytes of the 4 byte value returned are set to zero.

Example: The following example shows that the W@ operator will
return the 16-bit value which is stored starting as the
specified address :

HEX   <cr> ok <$0>
VARIABLE   STORAGE   <cr> ok <$0>
321F   STORAGE   W!   <cr> ok <$0>

↓STORAGE

Bytes in memory :

| | | | | | | | | | | 32 | 1F | | | | | | | | |

lower memory --> higher memory

0 0	0 0	3 2	1 F

STORAGE   W@   .   <cr> 321F ok <$0>

For Assembly Language Programmers:

```
      CODE  W@  ( a - w )
            MOVE.L    (A5)+,A0
            CLR.L     D1
            MOVE.W    (A0),D1
            MOVE.L    D1,-(A5)
            RTS
      END-CODE
      MACH
```

See also: W! , C@ , C! , @ , !

G-198

Memory
Operator

WHILE (f -)

Format:
```
BEGIN
 ( code executed each time though the loop )
flag
WHILE
 ( code executed while flag is true )
REPEAT
( code executed when loop is terminated )
```

Action: WHILE is the decision maker in the BEGIN...WHILE...REPEAT
program control structure. If the flag passed to WHILE is
true (non-zero), the code between the WHILE and REPEAT
will be executed until the REPEAT is reached. Upon reach-
ing the REPEAT, program execution will be rerouted back
to the code which follows the BEGIN. If the flag passed to
WHILE is false (zero), WHILE will terminate the loop by
reroutine program execution to to the code which immediately
follows the REPEAT. Due to the structure of the
BEGIN. . .WHILE. . .REPEAT loop, code between the
WHILE and REPEAT may not be executed at all if a false
flag is passed to WHILE during the first pass through the
loop.

Example:

```
: TEST-LOOP                    ( Code executed each time
  BEGIN                          through the loop. )
   ." Hit a space to continue" CR
  KEY 32 =
  WHILE
   ." I'm still alive !! " CR
  REPEAT ;
```

(Phrase which generates a flag for WHILE.)

(REPEAT will always reroute program execution to the code which follows the BEGIN.)

(Code which is executed only if flag passed to WHILE is true.)

See Also: BEGIN , REPEAT , UNTIL , AGAIN

G-199

Program
Control
Structure

WORD   (c - adr)

Format: ascii-value WORD

Action: Reads a string from the input stream using the given character as a delimiter. Once WORD has found a string surrounded by the given delimiter it inserts a length byte in front of the string to indicate how many characters are in that string. Then WORD moves the whole string including the length byte to a location 4 bytes above the top of the names. WORD puts the string at this location because if the string happens to be the name of a new definition, that new definition will already have its name in place and the rest of the dictionary entry may be built around it. WORD leaves the address of the length byte of the string on top of the stack.

WORD also has a special "wild-card" option. If a zero is passed to WORD it will return the next character in the current input stream.

Example: The example below will show how WORD may be used to get the next word surrounded by spaces from the input stream. The word will be stored in a storage location for future use:

```
VARIABLE  Storage  20  VALLOT  <cr> ok <0>
: GetString  { dest | len src - }
     32  WORD                    (Gets next string surrounded by spaces. )
     COUNT  ->  len              ( Get length byte and actual string address. )
              ->  src
     -1  +>  src                 ( Adjust address and length to include length )
     1  +>  len                  ( byte in CMOVE operation. )
     src  dest  count  CMOVE  ; ok <0> ( Move string to storage location. )
Storage  GetString  HELLO  <cr> ok <0>
Storage  COUNT  TYPE  <cr> HELLO ok <0>
```

For Advanced Programmers:

WORD normally passes all characters it receives through an internal "translate table". The translate table used by WORD does not affect the case of any character it receives but it does convert all non-printable characters (CR , LF , TAB , ESC , etc) to spaces.

The translate table is not used with the wild-card option mentioned above. When the end of the input stream is reached when using the wild-card option a zero-length string will be returned. The following example shows how the screen line comment operator, \ , may be defined by using the wild-card option :

```
: \  BEGIN  0  WORD  COUNT  0= SWAP  C@  13  =  OR  UNTIL  ; IMMEDIATE
```

\ ignores all characters until a zero length string or a carriage return is encountered.

See also: #TIB   G-200   Character I/O

! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z{|}~

WORDS

Format: WORDS

Action: Displays on the screen all words which belong to the vocabulary which is currently being searched first, the transient vocabulary. The listing may be suspended at any time by pressing a key and restarted again by hitting the space bar. Striking any other key will terminate a suspended WORDS listing.

Example: To see a listing of the FORTH vocabulary :

ONLY FORTH <cr> ok <0>

WORDS <cr>

TURNKEY	MAKEMODULE	TCALL	DUMP
.S	"	ASCII	ASSIGNMODULE
$	QUIT	\	(
DEPTH	ABORT	ABORT"	ABORT_VECTOR

Space Bar to Continue

ok <0>

To see a listing of the ASSEMBLER vocabulary :

ALSO ASSEMBLER <cr> ok <0>

WORDS <cr>

ADDX.L	ADDX.W	ADDX.B	SUBX.L
SUBX.W	SUBX.B	SBCD	ABCD
RTD	STOP	MOVEP.L	MOVEP.W
CMPM.L	CMPM.W	CMPM.B	MOVEM.W

Space Bar to Continue
ok <0>

See also: FORTH , ASSEMBLER

G-201

FORTH
Tool

W_EXT (n1 - n2)

" word-extend"

Format: 8-bit-value W_EXT

Action: Extends the byte-length (8-bit) value in the lower byte of
               the number on top of the stack into a word-length (16-bit)
               value by copying bit 7, the sign bit for a byte-length value,
               to bits 9-15 of the word-length value.

See also: L_EXT

G-202

Number I/O

XOR   (n1 n2 - n3)
" exclusive-or "

Format: n1 n2 XOR

Action: Performs the bit-by-bit logical exclusive-or of n1 with n2.
                 Leaves the result on top of the stack.

Example: The truth table for XOR is:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A common use of XOR is for flipping bits in a number. In the following example the binary
number 11111111 is used to flip all the bits (perform a one's complement) of a byte value :

BINARY   <cr> ok <%0>

10101010   <cr> ok <%1>

11111111   XOR   <cr> ok <%1>

. <cr> 1010101 ok <%0>

DECIMAL   <cr> ok <0>

For Assembly Language Programmers:

        CODE XOR (n1 n2 - n3)
            . MOVE.L   (A5)+,D0
             EOR.L   D0,(A5)
             RTS
        END-CODE
        MACH

See also: OR , AND , NOT

G-203

Logic
Operator

```
! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~
```

[_____

" left-bracket "

Format: Generally used in the following format -

                  . . . . . code being compiled . . . . .
                  [code to be executed immediately]
                  . . . . . code being compiled . . . . . .

Action: Ends compilation so that subsequent text is interpreted.

Example: : IMPATIENT ." Couldn't wait !" ; <cr> ok <0>

: PATIENT
    [IMPATIENT] ." I waited. " ; <cr> Couldn't wait ! ok <0>

(IMPATIENT was executed even though
  it is not an immediate word because
  [put the system into the interpreting
  state. Any words which follow [, such
  as IMPATIENT, are forced to execute
  immediately and will not compile any
  code.)

PATIENT <cr> I waited. ok <0>

(When PATIENT is executed, only the words which
  were allowed to compile code will be run.)

See also:] , IMMEDIATE , COMPILE

Compilation
Word

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

['] (- a)

Format: ['] <name>

Action: Immediate word used within a colon definition to compile the
               parameter field address of the next word in the definition as a
               literal. During execution of the definition the address will be
               pushed on the stack. If the word is not found in the current
               dictionary search an error message is issued.

               ['] is a smart word that will support both PC-relative and jump
               table addressing modes.

Example: ['] could be used to store the address of a custom
               ABORT handling routine in the ABORT_VECTOR
               system variable:

               : INSTALL ['] CustomAbort ABORT_VECTOR ! ;

               NOTE: ['] must be used in colon definitions.

See also: ' , LITERAL

G-205

Compilation
Word

`! " # S % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [ \ ] ^ _ ` a-z { | } ~`

[COMPILE]

" bracket-compile

Format: : <name> ..... [COMPILE] <immediate definition> ..... ;

Action: Forces the compilation of an IMMEDIATE definition which
               would normally be executed during compilation.

Example: The following sequence of words could be used in place
                  of [COMPILE] to achieve the same effect:

                  . . . ['] <name> EXECUTE . . . .

The immediate word SPEAK-NOW from the IMMEDIATE glossary entry can be compiled
by [COMPILE] :

: SPEAK-NOW ." Compiling..." ; IMMEDIATE <cr> ok <0>

: SPEAK-LATER [COMPILE] SPEAK-NOW ; <cr> ok <0>

SPEAK-LATER   <cr> Compiling... ok <0>

See also: IMMEDIATE , COMPILE

Compilation Word

G-206

\ _____

" back-slash "

Format: \

Action: Single-line commenting word. Any words following \
                 on the current line will be skipped over.

See also: (

G-207

FORTH
Tool

]
_____

" right-bracket "

Format:	Generally used in the following format -

> . . . . . code being compiled . . . . .
> [code to be executed immediately]
> . . . . . code being compiled . . . . . .

Action: Ends interpretation so that subsequent text is compiled.

<u>Example:</u>   Vectored execution :

HEX

: RTS 4E75 W, ; IMMEDIATE	(A 4E75 is the opcode for an RTS instruction.)
CREATE VECT-RUN	(Assembly language instructions generated :)
] U P	(JSR UP)
RTS	(RTS)
DOWN	(JSR DOWN)
RTS	(RTS)
LEFT	(JSR LEFT)
RTS	(RTS)
RIGHT	(JSR RIGHT)
RTS [	(RTS)
: VECTORED (n -)	
**6 **	(Multiply the index by 6 since a 'jump to subroutine' [JSR] plus an RTS instruction takes up a total of 6 bytes.)
VECT-RUN +	(Add the index-offset to the address of the start of the table.)
EXECUTE ;	(EXECUTE what is at this address.)
2 VECTORED <cr>	(This would cause LEFT to be executed.)

See also: [, IMMEDIATE , COMPILE

Compilation Word

G-208

^

Format: ^ <name of local variable or named input parameter>

Action: Returns the address of the local variable or named input
parameter whose name follows it. ^ must be used in a
colon definition.

Example: Local variables should be used whenever temporary
scratch variable space is needed. The word 'Get2Digits'
below uses EXPECT to get a 2 digit input from the user.
Since the input will be very small (a 2 digit input requires
only two bytes) a local variable location can be used for
the EXPECT buffer. By using a local variable rather
than a global variable (created by VARIABLE), four
bytes of variable space are saved and the routine
becomes re-entrant:

```
ONLY  FORTH  DEFINITIONS
DECIMAL

: Get2Digits {   |  ExpectBuf --   }
    CR
    ." Input a two digit number -> "
    ^ ExpectBuf 1+  2   EXPECT \ pass address of local variable
    ^ ExpectBuf  NUMBER?         \ to EXPECT and NUMBER?
    IF
          ." The number was -> "  .
    ELSE
          DROP
          CR ." Invalid Input. "
    THEN  ;

Get2Digits  <cr>
Input a 2 digit number -> 3 4   The number was -> 34
ok <0>
```

See also: -> , +>

                              System/Local
                              Variable

`! " # $ % & ' ( ) * + , - . / 0-9 : ; < = > ? @ A-Z [\]^_ `a-z {|}~`

{ _____

Format:	{ (names of initialized local variables) |
	(names of non-initialized local variables) -- (comments) }

Action: { marks the start of a local variable list. Local variables
are those whose contents are valid only when the particular
definition that they are defined for is being executed.
Because they are local variables, as opposed to global variables,
they will support reentrant code when writing recursive or
multi-tasking programs.
A local variable list may contain up to three fields--a field for
initialized local variables, a field for non-initialized local
variables, and a comment field. Examples of uses of and syntax
for local variables are given below.

Example: The following local variable example takes two numbers off the
stack, stores them in the named input parameters X and Y,
squares each number, and adds the squared values together
to get the final result.

```
: SQUARED  { X  Y  -- result  }
   X  X  *
   Y  Y  *
   +          ;  <cr> ok  <0>

4  5  TEST  .  <cr> 41 ok  <0>
```

Anything in the local variable list after a '--' is treated as a comment and ignored.
The name 'result' is included only as a reminder that a value is left on the stack.
Other equivalent local variable lists for the above example are :

```
{ X Y - }    { X Y | }   { X Y | - }
```

See also: ->

G-210

System/Local Variable

OS-9 VOCABULARY WORDS

$CLOSE
$CREATE
$DELETE
$OPEN
$WRITE
?OS9ERROR
ERRORPATH
FILEID
PARAM_PTR
RESPONSE

$CLOSE (pathnumber - errorcode)
$CLOSE will close the file specified by the path number.

$CREATE (pathenameaddr fileattributes accessmode
                                    - pathnumber errorcode)
$CREATE will try to create using the specified pathname and will try to give the file the
specified permission attributes. If successful, $CREATE will open the file with the specified
access mode and will return the pathnumber as the second number on the stack.
$CREATE will return an error code on top of the parameter stack.

$DELETE (pathnameaddr accessmode - errorcode)
$DELETE will look for the file specified by the name string and try to delete it.

$OPEN (pathnameaddr accessmode - pathnumber errorcode)
$OPEN will try to open a the file specified by the path string using the specified access mode.

$WRITE (bufferaddr len pathnumber - #byteswritten errorcode)
Takes the requested number of bytes from the buffer pointed to by the 'bufferaddr'
parameter and attempts to write them to the open file specified by the file reference
number. The bytes will be written to the file starting at the specified offset into the file.
After the write is completed the number of bytes actually written and an error code
will be returned on the stack.

?OS9ERROR   (errorcode -)

Evaluates the error code passed to it. If an error has occurred, ?OS9ERROR will print
out an error message in OS-9 format (Error #mmm:nnn). If ERRORPATH contains
a valid path number for an error message file, an additional text message will also be
printed. If no error has occurred, ?OS9ERROR will do nothing.

ERRORPATH   (-   a)

ERRORPATH is the MACH2 system variable used to hold either the path number of a
valid error message file or 0. (see ?OS9ERROR above)

FILEID   (- a)

FILEID is the MACH2 system variable used to hold the path number of the current path.

PARAM_PTR   (- a)

Returns the address of the null-terminated parameter string passed to this process
when the process was started.

RESPONSE   (- a)

RESPONSE is the MACH2 system variable used to hold the address of a signal
intercept handling routine.

MATH VOCABULARY WORDS

F!
F*
F+
F-
F.
F.S
F/
F=
F>I
F@
FATAN
FCOS
FDROP
FDUP
Fe^x
FIXED
Flog
Fln
FNEGATE
FOVER
FP
FPICK
FROLL
FROT
FSIN
FSQRT
FSWAP
FTAN
Fy^x
I>F
INT
PRECISION

F !   (a1 · | F1 ·)

Stores the floating point number on top of the floating point stack into the address on top of the parameter stack.

F *   (· | F1 F2 · F3)

MultipliesF1*F2 and leaves the floating point result on top of the floating point stack.

F +   (· | F1 F2 · F3)

Replaces the two numbers on top of the floating point stack with their floating point sum.

F-   (· | F1 F2 · F3)

Subtracts F1-F2 and leaves the floating point result on top of the floating point stack.

F .   (· | F ·)

Displays the number on top of the floating point stack on the screen according to the current display characteristics set by FIXED and FLOAT.

F . S   (· | ·)

Non-destructively displays the contents of the floating point stack.

F /   (· | F1 F2 · F?)

Divides F1/F2 and leaves the floating point result on top of the floating point stack.

F =   (· f | F1 F2 ·)

Compares the two numbers on top of the floating point stack. Returns a true (non-zero) flag on the parameter stack if the two numbers are equal. Returns a false (zero) flag on the parameter stack if the two numbers are not equal.

F > I   (· n | F ·)

Converts the number on top of the floating point stack to an integer and puts it on top of the parameter stack.

F @   (a · | · F)

Puts the floating point number stored at the address on top of the parameter stack on top of the floating point stack.

FATAN   (· | F1 · F2)

Calculates the arctangent of the number on top of the floating point stack and leaves the result on top of the floating point stack.

SANE Vocabulary Words

FCOS   (- | F1 - F2)

Replaces the angle on top of the floating point stack by the cosine of the angle.
The angle should be expressed in radians.

FDROP   (- | F1 -)

Removes the number on top of the floating point stack.

FDUP   (- | F - F F)

Duplicates the number on top of the floating point stack

Fe^x   (- | F1 - F2)

Calculates the natural or base-e exponential of the number on top of the floating point
stack and returns the result on top of the floating point stack.

FIXED   (n - | -)

Uses the number on top of the parameter stack to set the number of digits which will be
displayed after the decimal point in FIXED point format. After FIXED is used, all floating-
point numbers will be displayed in fixed point format with the specified number of digits
displayed after the decimal point. The default fixed number display includes four digits
after the decimal point.

Fln   (- | F1 - F2)

Calculates the natural or base-e logarithm of the number on top of the floating point stack
and returns the result on top of the floating point stack.

Flog   (- | F1 - F2)

Calculates the logarithm to the base 10 of the number on top of the floating point stack and
returns the result on top of the floating point stack.

FNEGATE   (- | F - -F)

Negates the value of the floating point number on top of the floating point stack.

FOVER   (- | F1 F2 - F1 F2 F1)

Puts a copy of the second number on the floating point stack on top of the floating point stack.

F P   (- | -)

Puts the system in the floating point mode. All numbers entered after FP which contain
decimal points (periods) or exponents will be converted to 80-bit floating point numbers and
placed on the floating point stack. Numbers which do not contain decimal points will be
treated as integer values and will be placed on the parameter stack. INT returns the system
to the integer mode.

SANE Vocabulary Words

FPICK (n - | - F)

Moves a copy of the nth item down on the floating point stack (where n = 0 refers to the top item on the floating point stack) to the top of the floating point stack. FPICK takes the integer value n from the top of the parameter stack.

FROLL (n - | -)

Rotates the nth item on the floating point stack to the top of the floating point stack (where n=0 refers to the item on top of the floating point stack). FROLL takes the integer value n from the top of the parameter stack. N must be greater than 0.

FROT (- | F1 F2 F3 - F2 F3 F1)

Rotates the third number on the floating point stack to the top of the floating point stack.

FSIN (- | F1 - F2)

Replaces the angle on top of the floating point stack by the sine of the angle. The angle should be expressed in radians.

FSQRT (- | F1 - F2)

Replaces the number on top of the floating point stack with its square root.

FSWAP (- | F1 F2 - F2 F1)

Switches the positions of the two numbers on top of the floating point stack.

FTAN (- | F1 - F2)

Replaces the angle on top of the floating point stack by the tangent of the angle. The angle should be expressed in radians.

Fy^x (- | F1 F2 - F3)

Calculates the value F1^F2 using the two numbers on top of the floating point stack and returns the result on top of the floating point stack.

I>F (n - | - F)

Converts the number on top of the stack to a floating point number and puts it on top of the floating point stack.

INT (- | -)

Puts the system into integer mode. All numbers entered are treated as integers and placed on the parameter stack.

PRECISION (n1 n2 - | -)

Sets the precision for all subsequent floating point operations. The two values passed to PRECISION are used to specify the precision. The bottom table on page 12-4 of the OS-9/68000 Operating System Technical Manual contains the hexadecimal representations which should be passed to PRECISION.

SANE Vocabulary Words

ASSEMBLER DIRECTIVES

.ALIGN
DC
DS
EQU
HEADER
LABEL
OS9

.ALIGN (-)

.ALIGN is used to ensure that the next piece of data (either code or constant values) put into the dictionary will be aligned in memory on an even word boundary. If the next position in the dictionary lies on an odd address a zero byte will be inserted as 'padding'. .ALIGN is commonly used when defining character strings since string data often ends up on an odd address boundary:

```
HEADER  Error1      DC.B    12,'Invalid  Year'
                    .ALIGN
HEADER  Error2      DC.B    13,'Invalid  Month'
                    .ALIGN
HEADER  Error3      DC.B    11,'Invalid  Day'
                    .ALIGN
HEADER  Error4      DC.B    12,'Invalid  Hour'
                    .ALIGN
```

DC (-)   Format: HEADER <name> DC.s <data>

DC is used to place constant data into the dictionary area. <data> can be an arithmetic expression, a character string, or any number of expressions or character strings separated by commas. A character string may contain any printable ASCII character (including spaces) and must be surrounded by single quotes. To include a single quote in a character string, precede the single quote by a single quote. The size suffix (L, W, or B) is used to specify the size of the data laid down by DC. If DC.W or DC.L is used to generate a character string constant the final word or long word will be padded (to the right) with zeroes if the characters do not completely fill it. HEADER is used with DC to mark the position of the data in the dictionary:

```
HEADER  LongData       DC.L    $434F4445
HEADER  ExpressionData DC.W    45+(5*6)-(4/2)
HEADER  LotsOfData     DC.B    45*3,$7F,'Hi','Bye'
                       .ALIGN
HEADER  StringData     DC.B    'String  with  spaces'
                       .ALIGN
```

DS (-) Format: LABEL <name> DS.s <num>

DS is used to reserve variable storage space in the data area. <num> is an arithmetic
expression or constant used to indicate how many 'units' of memory should be reserved.
The size of a 'unit' is specified by the DS suffix (L, W, or B). LABEL is used to mark the
location of the reserved variable space in the data area. Each of the uses of DS below will
cause 48 bytes of variable storage to be reserved:

```
LABEL   MethodOne    DS.L    $C
LABEL   MethodTwo    DS.W    24
LABEL   MethodThree  DS.B    48
LABEL   MethodFour   DS.B    2*2*4*(3+12/4)
```

EQU (-) Format: EQU <name> <value>

Assembler directive used to assign values to a symbolic name. The value assigned to
the name may be a constant,expression, or addressing mode.

Assigning an expression to a name (the expression will be evaluated according to the arithmetic precedence rules described in the assembler section):	EQU Secs/Year	365*24*60*60
Assigning a constant (number) to a name:	EQU CR $13	
Assigning an addressing mode to a symbolic name:	EQU M$Name	$C(A0)

HEADER (-) Format: HEADER <name> DC(B).X <values>

Used to mark the location of constant data. The address of constant data may be
obtained by 'ticking' the name of its HEADER.

LABEL (-) Format: LABEL <name> DS.X <number>

Used to mark the location of variable storage space. The address of a storage space
may be obtained by executing the name following LABEL.

OS-9 (-) Format: OS9 <name of system routine>

Assembler word used to generate OS-9 system calls.

Appendices

APPENDIX A: MACH 2 Dictionary Header Structure

Dictionary Header:

		Link Field (4 bytes)
LENGTH BYTE	C	Length Field (1 byte)
A	T	Name Field (variable length)
		Segment Field (2 bytes)
		Parameter Field Pointer (4 bytes)

Total header field length is 11 bytes plus the length of the name field.

The LINK FIELD contains a relative 32-bit pointer to previous word in the dictionary which is also in the same vocabulary.

The SEGMENT FIELD contains a value which indicates how the parameter field address may be found. The SEGMENT FIELD works together with the PARAMETER FIELD:

SEGMENT FIELD	MEANING	PARAMETER FIELD
0	User-defined word.	Contains offset to word from base of kernel.
1	Kernel word.	Contains jump table offset.
2	Floating point word.	Contains jump table offset.
3	Disk I/O word.	Contains jump table offset.
4	Compiler word	Contains jump table offset.

Close-Up of Length Byte:

7   0

[ ][ ][ ][ ][ ][ ][ ][ ]   Length Field   (1byte)

Name length. Names containing up
to 32 characters may be used.

SMUDGE bit

MACH bit

IMMEDIATE bit

A-2

Dictionary Header Structure

APPENDIX B: MACH 2 Register Usage

A7   The A7 is the register used to control MACH 2's subroutine stack. This is the stack which holds all the return addresses generated when MACH 2 jumps to subroutines.

A6   OS-9 uses the A6 register to hold the address of the top of a module's data area. MACH 2 keeps its subroutine stack, block buffers, jump tables, and variable space in the data area. Use of the A6 register is strongly discouraged.

A5   The A5 is the register used to point to the MACH 2's parameter stack. The A5 and A7 stacks grow downward in memory. Only long word values may be put onto the A5 stack. The A7 stack may receive word or long word values.

A4   The A4 register is used as a linkage register for MACH 2's local variables. This register may be used by a program which does not use local variables.

A3   The A3 register is used in conjunction with the D5 and D6 registers to control MACH 2's loop return stack. This is the stack used by DO loops. Only long word values are put onto the A3 stack.

A2   Untouched by MACH2.

A1   MACH2 scratch registers.

A0

D7   The D7 is used for the floating point stack. This is the stack used by all floating point operators. The D7 register may be used by a program which does not use any floating point routines.

D6   The D6 register holds the top item on the return stack. During DO...LOOPs the D6 register will be holding the current index value for the loop.

D5   The D5 register holds the second item on the return stack. During DO...LOOPs the D5 register will be holding the limit value for the loop.

D4   Untouched by MACH2.

D3   **D1**   MACH2 scratch registers.

D2   **D0**

A-3

MACH 2 Register Usage

APPENDIX C: THE MACH 2 LOOP (RETURN) STACK

Most FORTH's use the system stack for their return stack. This means that any definitions which use the return stack must make sure that all values they place on the return stack are removed before the definition terminates. Such a return stack can only be used for temporary storage during the execution of the definition. ANY parameters left on the system stack would cover up important subroutine return addresses and would cause a fatal system crash.

In MACH 2 a completely separate stack is available as a return stack. To avoid confusion between the subroutine return stack and the FORTH return stack, this separate stack is called the 'Loop Stack' since it is primarily used by FORTH to hold the loop indicies used during DO...LOOPS. The loop stack is also available as a general purpose stack for use in your programs.

The loop stack has been implemented differently than a normal stack to speed up the execution of the DO...LOOP control structure. If you use FORTH's return stack operators you don't have to worry about the details of the special loop stack implementation. If you are programming in assembly language or if you are trying to disassemble programs which use DO...LOOPS, please read on.

How the loop stack is implemented

First, a diagram of how the loop stack acts when an item is added to it:

BEFORE:

A3: `0 0 0 2 5 6 4 4` stack pointer

D6: `0 0 0 0 0 0 5 6` top item

D5: `0 0 0 0 0 0 2 4` second item

AFTER

A3: `0 0 0 2 5 6 4 8` stack pointer

D6: `0 0 0 0 0 5 4 3` top item

D5: `0 0 0 0 0 0 5 6` second item

00025644

third item

'base'
of the
stack →

Items
on the
stack

00025648

00000024   third item

'base'
of the
stack →

Items
on the
stack

A-4

Loop Stack

The first thing to notice is that the top two items on the loop stack are kept in the D6 and D5 registers. The third and following items are kept on a stack whose stack pointer is kept in the A3 register. Also note that the actual 'stack' portion of the loop stack is an upward growing stack (normally stacks grow downward in memory).

It takes three steps to add an item to the loop stack:

1. First the second item on the stack (located in the D5 register) must be moved to the 'A3 stack'. The item is stored at the address contained in the A3 register and then the address is incremented by four bytes. This means the stack pointer is always left pointing at the next available slot in the stack.

2. Second, the top item on the stack (located in the D6 register) is placed in the second slot on the stack (in the D5 register).

3. Then the new item is put in the top loop stack position (in the D6 register).

The return stack grows up and down through the D5 and D6 registers.

The reason this return stack implementation speeds up the DO...LOOP control structure is that during the execution of a DO...LOOP, the limit and index will always be the top two items on the stack. With this return stack the top two items are always kept in registers. If you were to study the 68000 instruction execution times you would see that instructions which operate on registers execute much faster than instructions which operate on the contents of memory locations.

A-5

Loop Stack

APPENDIX D: Subroutine Threading

General Discussion

MACH 2 is subroutine-threaded code. This means that words like 'FORGET' or 'ROT' are actual assembly-language subroutines used by other subroutines such as 'QUERY' or 'WORDS'. The 68000 runs everything.

While this may sound only natural, such threading is actually new to FORTH. Older processors, such as the 8085, were not equipped to handle the stack-oriented structure of FORTH so a virtual CPU had to be written that could keep track of the stacks. This led to a language that ran via lists of addresses of routines ('address-threading') which in turn were either lists of addresses or the final assembly language to be executed. The speed penalty of this virtual CPU is obvious.

The 68000, however, is perfectly equipped to run FORTH by itself due to its general-purpose registers and addressing-modes which allow many stacks to exist. The old NEXT instruction, the brain of the virtual CPU, has been replaced by the 68000's own RTS instruction which executes in 2 μs (that's two millionth's of a second). The switch to subroutine-threading brings about tripling in execution speed. It does also cause a slight increase in the size of the code generated but, in today's large memory machines, program size is becoming less of an issue. In addition, the language becomes much more understandable; there's less magic.

The bottom line? FORTH is now three times as fast but still retains its traditionally-small kernel. It's still an interactive development language whose programs are written and debugged in perhaps 1/4 the time required for an edit-compile-try again language like C.

How Subroutine-Threading Is Implemented

Very simply, references to other Forth definitions are compiled as JSR's (that's a 68000 mnemonic for Jump to Subroutine). At run-time the 68000 jumps to this subroutine and returns when the subroutine has finished execution. For instance, this is the assembly language code for the FORTH word U. :

```
'  U.  5  IL
U.
089F86:       JSR       $-8A(PC)     ; $89EFC     <#
089F8A:       JSR       $-14(PC)     ; $89F76     #S
089F8E:       JSR       $-80(PC)     ; $89F0E     #>
089F92:       JSR       $-9C(PC)     ; $89EF6     TYPE
089F96:       JSR       $-70(PC)     ; $8A006     SPACE
 ok <0>
```

U. is comprised of a series of JSR's to other FORTH words. All words in the FORTH kernel reference each other via PC-relative JSR's.

A-6

Subroutine Threading

APPENDIX E: MACRO SUBSTITUTION

Macro substitution is a technique used at compile time to increase speed by laying in the actual code for a routine instead of compiling a jump to the subroutine. For example, in the definition

```
: SQUARED    DUP  *  ;
```

the compiler could lay in a jump (4 bytes) to the 'DUP' subroutine (JSR) and then a jump to the '*' subroutine. The code for 'DUP', however is only MOVE.L (A5),-(A5), a two-byte instruction that replicates what is on the parameter stack. By laying in the code for 'DUP' instead of a jump to the 'DUP' subroutine, 'DUP' runs over three times faster. The code for '*', though, is quite large. In this case, the tiny increase in speed does not justify using such a large amount of memory. The compiler will compile a PC-relative jump to the '*' subroutine instead. A PC-relative jump-to-subroutine instruction (JSR d(PC)) requires 4 bytes.

Using Macro Substitution

A FORTH word is marked as a macro by setting the MACH bit in the name field of a definition. Usually the word MACH is used for this purpose. MACH is used in the same manner as the word 'IMMEDIATE'. When the compiler encounters a word with its MACH bit set, it knows it should lay the code for the word, 'DUP' for example, IN the definition instead of a JSR to it. The compiler will move the code from the beginning of the routine up to, but not including, the first RTS ($4E75) it finds, into the definition being compiled.

Precautions

Do not use any PC-relative references to words outside of the current definition in words to be 'macro-ed'. A PC-relative reference is not valid once it has been moved.

To transfer correctly, MACH routines should have ONLY one exit point (RTS) and that exit should be located in the last line of the routine.

Caution, excessive use of MACH words may yield a large increase in program size with only a small speed improvement.

Examples

The word '!' is defined as follows:

```
        CODE !      ( n  a  - )
            MOVE.L    (A5)+,A0
            MOVE.L    (A5)+,(A0)
        RTS
        END-CODE    MACH
```

The following disassembly shows how ! is compiled into a definition, along with several
several other characteristics of compilation. The word to be disassembled is the
following (DUMMY is a variable):

```
    : TEST  0  2  DUP  ROT  DUMMY  !  ;
```

```
MOVEQ.L  #$0,D0
MOVE.L   D0,-(A5)          Move a 0 on to the stack (4 bytes).
MOVEQ.L  #$2,D0
MOVE.L   D0,-(A5)          Move a 2 on to the stack (4 bytes).
MOVE.L   (A5),-(A5)        Make another copy of whatever's on the stack (4 bytes).
JSR      $-7BE6(A6)        Jump through the jump table to the code for ROT (4 bytes).
LEA      $-71F8(A6),A0     Get address of DUMMY into a register (4 bytes).
MOVE.L   (A5)+,(A0)        Store top stack item into variable location (2 bytes).
RTS                        Return to whichever word called this one (2 bytes).
```

There are four types of compilation apparent in this example. Numbers are compiled,
not as FORTH literals, but as 68000 MOVE instructions. DUP and ! are marked as
macro words so their code is laid directly into the definition. References to non-macro
FORTH words (such as ROT) are compiled as JSR's through the jump table to the proper
subroutines.

A-8

Macro Substitution

APPENDIX F: Suggested Reference Readings

FORTH

Starting FORTH by Leo Brodie
c 1981 by FORTH, Inc., Hermosa Beach, CA 90254
Prentice-Hall, Inc.
Beginners book on FORTH.

Thinking FORTH by Leo Brodie
c 1984
Prentice-Hall, Inc.
Discusses FORTH style and programming techniques.

68000 Assembly Language

M68000 16/32-Bit Microprocessor Programmer's Reference Manual, 4th Edition
c 1984 by Motorola Inc.
Prentice-Hall, Inc.
Describes the 68000 processor and each 68000 assembly language instruction.

68000 Assembly Language Programming
by Gerry Kane, Doug Hawkins, and Lance Leventhal
c 1981 by McGraw-Hill, Inc.
OSBORNE/McGraw-Hill
 isses 68000 assembly language programming for the beginner.

OS-9 Programming

OS-9/68000 Operating System Technical Manual
OS-9/68000 Macro Assembler User's Manual
OS-9/68000 Operating System User's Manual

Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322
(515) 224-1929

APPENDIX G: MACH 2 Error Messages

?   Name not found.

ASCII character is missing   The word ASCII did not find an ascii character to
                                    convert to a number.

Can't   An attempt has been made to FORGET a protected
                                    word.

Compile Only !   An attempt has been made to immediately execute
                                    a word which may only be used within a colon definition.

Dictionary entry not specified   A defining word (word which creates and names new
                                    dictionary entries) was executed, but the name to
                                    be given to the new dictionary entry did not follow the
                                    defining word on the same line. Typing VARIABLE <cr>
                                    would generate this message.

Divide by Zero Error !   Division with a divisor of zero was attempted.

Empty   Message issued by the word .S when the stack is
                                    empty and there are no numbers for it to display.

Illegal Instruction at   The 68000 encountered an instruction it did not
                                    recognize at the address indicated.

is Redefined   A new definition has been created with the same name
                                    as a currently existing definition. Because dictionary
                                    searches will terminate at the first version of a word
                                    found, the previously defined word is now hidden.

Line 1111 error at   A Line 1111 exception (usually a breakpoint) has
                                    been encountered at the specified address.

Missing   )   A comment is missing its right parenthesis.

Missing WHILE   The WHILE is missing from a BEGIN..WHILE..
                                    REPEAT loop.

No Block Zero !   The FORTH-83 standard does not allow LISTing
                                    or LOADing of block 0.

Return stack misalignment	The return stack pointer changed position during the compilation of a definition. This can be caused by leaving extra values on the return stack or by removing too many values from the return stack during the compilation process. This error condition generates an abort.
Single character only	A string with more than 1 character followed the word ASCII.
Stack Empty !	An attempt was made to take a number from the parameter stack when the stack was empty.
Unpaired BEGIN	A BEGIN is missing its corresponding AGAIN, UNTIL, or REPEAT.
Unpaired CASE	A CASE is missing its corresponding ENDCASE.
Unpaired DO	A DO is missing its corresponding LOOP or +LOOP.
Unpaired IF	An IF is missing its corresponding THEN.
Unpaired OF	An OF is missing its corresponding ENDOF.

MACH 2 Error Messages

APPENDIX H: OS-9/68000 USER MODE SYSTEM CALLS

F$AllBit   Sets bits in an allocation bit map.

      Assembler Call: OS9 F$AllBit

      Input: D0.W = Bit number of first bit to set.
                              D1.W = Bit count (number of bits to set).
                              (A0)   = Base address of an allocation bit map.

      Output: None.

      Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

F$Chain   Load and execute a new primary module

      Assembler Call: OS9 F$Chain

      Input: D0.W = Desired module type/language
                                                  (must be program/object or 0=any)
                              D1.L = additional memory size
                              D2.L = parameter size
                              D3.W = number of I/O paths to copy
                              D4.W = priority
                              (A0)   = module name ptr
                              (A1)   = parameter ptr

      Output: D0.W = child process ID
                              (A0)   = updated beyond module name

      Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

      Possible Errors: E$NEMod

F$CmpNam Compare two names.

      Assembler Call: OS9 F$CmpNam

      Input: D1.W = Length of pattern string
                              (A0)   = Pointer to pattern string
                              (A1)   = Pointer to target string

      Output: cc   = Carry bit clear if the strings match

      Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code if unequal or error

      Possible Errors: E$Differ , E$StkOvf

OS-9 User Mode System Calls

F$CpyMem Copy external memory.

        Assembler Call: OS9 F$CpyMem

        Input: D0.W   = Process ID of external memory's owner
                            D1.L   = Number of bytes to copy
                            (A0)   = Address of memory in external process to copy
                            (A1)   = caller's destination buffer pointer

        Output: None.

        Error Output: cc   = Carry bit set.
                            D1.W   = Appropriate error code.

F$CRC   Generate CRC.

        Assembler Call: OS9 F$CRC

        Input: D0.L   = Data byte count
                              D1.L   = CRC accumulator
                            (A0)   = Pointer to data

        Output: D1.L   = Updated CRC accumulator

        Error Output: cc   = Carry bit set.
                            D1.W   = Appropriate error code.

F$DatMod Create data module.

        Assembler Call: OS9 F$DatMod

        Input: D0.L   = Size of data required (not including header or CRC)
                              D1.W   = Module attr/revision
                              D2.W   = Module access permission
                              (A0)   = Module name string ptr

        Output: D0.W   = Module type/language
                              D1.W   = Module attr/revision
                              (A0)   = Updated name string ptr
                              (A1)   = Module data ptr ('execution' entry)
                              (A2)   = Module header ptr

        Error Output: cc   = Carry bit set.
                            D1.W   = Appropriate error code.

        Possible Errors: E$MNam

OS-9 User Mode System Calls

F$DelBit   Deallocate in a bit map.

> **Assembler Call:**   OS9 F$DelBit
>
> **Input:**   D0.W   ▪ Bit number of first bit to clear
> D1.W   ▪ Bit count (number of bits to clear)
> (A0)   ▪ Base address of an allocation bit map
>
> **Output:**   None.
>
> **Error Output:**   cc   ▪ Carry bit set.
> D1.W   ▪ Appropriate error code.

F$DExec   Execute debugged program.

> **Assembler Call:**   OS9 F$DExec
>
> **Input:**   D0.W   ▪ Process ID of child to execute
> D1.L   ▪ Number of instructions to execute (0=continuous)
> D2.W   ▪ Number of breakpoints in list
> (A0)   ▪ Breakpoint list
> register buffer contains child register image
>
> **Output:**   D0.L   ▪ Total number of instructions executed so far
> D1.L   ▪ Remaining count not executed
> D2.W   ▪ Exception occurred, if non-zero; exception offset
> D3.W   ▪ Classification word (addr or bus trap only)
> D4.L   ▪ Access address (addr or bus trap only)
> D5.W   ▪ Instruction register (addr or bus trap only)
> register buffer updated
>
> **Error Output:**   cc   ▪ Carry bit set.
> D1.W   ▪ Appropriate error code.
>
> **Possible Errors:**   E$IPrcID , E$PrcAbt

F$DExit   Exit debugged program.

> **Assembler Call:**   OS9 F$DExit
>
> **Input:**   D0.W   ▪ Process ID of child to terminate
>
> **Output:**   None.
>
> **Error Output:**   cc   ▪ Carry bit set.
> D1.W   ▪ Appropriate error code.
>
> **Possible Errors:**   E$IPrcID

A-14

OS-9 User Mode System Calls

F$DFork   Fork process under control of debugger.

          Assembler Call: OS9 F$DFork

          Input:

D0.W	=	Desired module type/revision (0=any)
D1.L	=	Additional stack space to allocate (if any)
D2.L	=	Parameter size
D3.W	=	Number of I/O paths for child to inherit
D4.W	=	Module priority
(A0)	=	Module name ptr (or pathlist)
(A1)	=	Parameter ptr
(A2)	=	Register buffer: copy of child's (D0-D7/A0-A7/SR/PC)

          Output:

D0.W	=	Child process ID
(A0)	=	Updated past module name string
(A2)	=	Initial image of the child process' registers in buffer

          Error Output:

cc	=	Carry bit set.
D1.W	=	Appropriate error code.

F$Exit   Terminate the calling process.

          Assembler Call: OS9 F$Exit

          Input: D0.W   = Status code to be retruned to the parent process

          Output: Process is terminated

          Error Output: None.

F$Fork   Create a new process.

          Assembler Call: OS9 F$Fork

          Input:

D0.W	=	Desired module type/revision (usually program/object 0=any)
D1.L	=	Additional memory size
D2.L	=	Parameter size
D3.W	=	Number of I/O paths to copy
D4.W	=	Priority
(A0)	=	Module name ptr
(A1)	=	Parameter ptr

          Output:

D0.W	=	Child process ID
(A0)	=	Updated past module name string

          Error Output:

cc	=	Carry bit set.
D1.W	=	Appropriate error code.

          Possible Errors: E$NEMod

OS-9 User Mode System Calls

F$GModDr Get module directory.

    Assembler Call: OS9 F$GModDr

    Input: D1.L   = Maximum number of bytes to copy
                          (A0)   = Buffer pointer

    Output: D1.L   = Actual number of bytes copied

    Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

F$GPrDBT Get process descriptor block table copy.

    Assembler Call: OS9 F$GPrDBT

    Input: D1.L   = Maximum number of bytes to copy
                          (A0)   = Buffer pointer

    Output: D1.L   = Actual number of bytes copied

    Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

F$GPrDsc Get process descriptor copy.

    Assembler Call: OS9 F$GPrDsc

    Input: D0.W   = Requested process ID
                          D1.W = Number of bytes to copy
                          (A0)   = Process descriptor buffer pointer

    Output: None.

    Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

    Possible Errors: E$PrcID

F$ID Get process ID / user ID.

    Assembler Call: OS9 F$ID

    Input: None.

    Output: D0.W   = Current process ID
                          D1.L   = Current process group/user number
                          D2.W = Current process priority

    Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

F$Icpt   Set up a signal intercept trap.

        Assembler Call: OS9 F$Icpt

        Input: (A0)   = Address of the intercept routine
                              (A6)   = Address to be passed to the intercept routine

        Output: Signals sent to the process will cause the intercept routine
                              to be called instead of the process being killed.

        Error Output: None.

F$Julian   Get Julian date.

        Assembler Call: OS9 F$Julian

        Input: D0.L   = Time (00hhmmss)
                              D1.L   = Date (yyyymmdd)

        Output: D0.L   = Time (seconds since midnight)
                              D1.L   = Julian date

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

F$Link   Link to memory module.

        Assembler Call: OS9 F$Link

        Input: D0.W = Desired module type/language byte (0=any)
                              (A0)   = Module name string pointer

        Output: D0.W = Actual module type/language
                              D1.W = Module attributes/revision level
                              (A0)   = Updated past the module name
                              (A1)   = Module execution entry point
                              (A2)   = Module pointer

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

        Possible Errors: EMNF , EBNam , E$ModBsy

A-17

OS-9 User Mode System Calls

F$Load   Load module(s) from a file.

        Assembler Call: OS9 F$Load

        Input: D0.B   ■ Access mode
                          (A0)   ■ Path name pointer

        Output: D0.W   ■ Actual module type/language
                          D1.W   ■ Attributes/revision level
                          (A0)   ■ Updated beyond path name
                          (A1)   ■ Module execution entry pointer (of 1st module loaded)
                          (A2)   ■ Module pointer

        Error Output: cc   ■ Carry bit set.
                          D1.W ■ Appropriate error code.

        Possible Errors: E$MemFul , E$BMID

F$Mem   Resize data memory area.

        Assembler Call: OS9 F$Mem

        Input: D0.L   ■ Desired new memory size in bytes.

        Output: D0.L   ■ Actual size of new memory area in bytes
                          (A1)   ■ Pointer to new end of data segment (+1)

        Error Output: cc   ■ Carry bit set.
                          D1.W ■ Appropriate error code.

        Possible Errors: E$DelSP , E$MemFul , E$NoRAM

F$PErr   Print error message.

        Assembler Call: OS9 F$PErr

        Input: D0.W   ■ Error message path number (0=none)
                          D1.W   ■ Error number

        Output: None.

        Error Output: None.

F$PrsNam   Parse a path name.

        Assembler Call: OS9 F$PrsNam

        Input: (A0)   ■ Name of string pointer

        Output: D0.B   ■ Pathlist delimiter
                          D1.W   ■ Length of pathlist element
                          (A0)   ■ Pathlist ptr updated past the optional "/" character
                          (A1)   ■ Address of the last character of the name +1

        Error Output: cc   ■ Carry bit set.
                          D1.W ■ Appropriate error code.

OS-9 User Mode System Calls

F$RTE   Return from interrupt exception.

      Assembler Call: OS9 F$RTE

      Input: None.

      Output: None.

F$SchBit   Search bit map for a free area.

      Assembler Call: OS9 F$SchBit

      Input: D0.W = Beginning bit number to search
                           D1.W = Number of bits needed
                           (A0)   = Bit map pointer
                           (A1)   = End of bit map (+1) pointer

      Output: D0.W = Beginning bit number found
                           D1.W = Number of bits found

      Error Output: cc   = Carry bit set.
                           D1.W = Appropriate error code.

F$Send   Send a signal to another process.

      Assembler Call: OS9 F$Send

      Input: D0.W = Intended receiver's process ID number (0=all)
                           D1.W = Signal code to send

      Output: None.

      Error Output: cc   = Carry bit set.
                           D1.W = Appropriate error code.

      Possible Errors: E$IPrcID , E$USigP

F$SetCRC   Generate valid CRC in module.

      Assembler Call: OS9 F$SetCRC

      Input: (A0)   = Module pointer

      Output: None.

      Error Output: cc   = Carry bit set.
                           D1.W = Appropriate error code.

      Possible Errors: E$BMID

F$SetSys Set/Examine OS-9 system global variables.

Assembler Call:	OS9	F$SetSys	
Input:	D0.W	=	Offset of system global variable to set/examine
	D1.L	=	Size of variable in least significant word (1, 2, of 4 bytes). The most significant bit, if set, indicates an examination request. Otherwise, the variable is changed to the value in register D2.
	D2.L	=	New value (if change request)
Output:	D2.L	=	Original value of system global variable
Error Output:	cc	=	Carry bit set.
	D1.W	=	Appropriate error code.

F$Sleep Put calling process to sleep.

Assembler Call:	OS9	F$Sleep	
Input:	D0.L	=	Ticks/seconds (number of ticks to sleep)
Output:	D0.W	=	Remaining number of ticks if awakened prematurely
Error Output:	cc	=	Carry bit set.
	D1.W	=	Appropriate error code.
Possible Errors:	E$NoClk		

F$SPrior Set process priority.

Assembler Call:	OS9	F$SPrior	
Input:	D0.W	=	Process ID number
	D1.W	=	Desired process priority: 65535=highest, 0=lowest
Output:	None.		
Error Output:	cc	=	Carry bit set.
	D1.W	=	Appropriate error code.
Possible Errors:	E$IPrcID		

F$SRqMem System memory request.

Assembler Call:	OS9	F$SRqMem	
Input:	D0.L	=	Byte count of requested memory
Output:	D0.L	=	Byte count of memory granted
	(A2)	=	Pointer to memory block allocated
Error Output:	cc	=	Carry bit set.
	D1.W	=	Appropriate error code.
Possible Errors:	E$MemFul , E$NoRAM		

A-20

OS-9 User Mode System Calls

F$SRtMem Return system memory.

        Assembler Call: OS9 F$SRtMem

        Input: D0.L   = Byte count of memory being returned
                             (A2)   = Address of memory block being returned.

        Output: None.

        Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

        Possible Errors: E$BPAddr

F$SSpd Suspend process.(currently not implemented).

        Assembler Call: OS9 F$SSpd

        Input: D0.W = Process ID to suspend

        Output: None.

        Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

        Possible Errors: E$NoClk

F$STime Set system date and time.   -

        Assembler Call: OS9 F$STime

        Input: D0.L   = Current time (00hhmmss)
                             D1.L   = Current date (yyyymmdd)

        Output: Time/date is set.

        Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

F$STrap Set error trap handler.

        Assembler Call: OS9 F$STrap

        Input: (A0)   = Stack to use if exception occurs
                                      (or zero to use the current stack)
                           (A1)   = Pointer to service request initialization table

        Output: None.

        Error Output: cc   = Carry bit set.
                             D1.W = Appropriate error code.

F$SUser   Set user ID number.

          Assembler Call: OS9 F$SUser

          Input: D1.L   = Desired group/user ID number

          Output: None.

          Error Output: cc   = Carry bit set.
                                  D1.W   = Appropriate error code.

F$SysDbg Call system debugger.

          Assembler Call: OS9 F$SysDbg

          Input: None.

          Output: None.

          Error Output: cc   = Carry bit set.
                                  D1.W   = Appropriate error code.

F$TIme   Get system date and time.

          Assembler Call: OS9 F$Time

          Input: D0.W   = Format: 0=Gregorian; 1=Julian;
                                              2=Gregorian with ticks; 3=Julian with ticks

          Output: D0.L   = Current time
                                  D1.L   = Current date
                                  D2.W   = Day of week (0=Sunday to 6=Saturday)
                                  D3.L   = Tick rate/current tick (if requested)

          Error Output: cc   = Carry bit set.
                                  D1.W   = Appropriate error code.

F$TLInk   Install user trap handling routine.

          Assembler Call: OS9 F$TLink

          Input: D0.W   = User trap number (1-15)
                                  D1.L   = Optional memory override
                                  (A0)   = Module name pointer
                                  If (A0)=0 or if [(A0)]=0, trap handler is unlinked.
                                  Other parameters may be required for specific trap handlers.

          Output: (A0)   = Updated past module name
                                  (A1)   = Trap library execution entry point
                                  (A2)   = Trap module pointer
                                  Other values may be returned by specific trap handlers.

          Error Output: cc   = Carry bit set.
                                  D1.W   = Appropriate error code.

A-22

OS-9 User Mode System Calls

F$UnLInk   Unlink a module by address.

Assembler Call:	OS9 F$UnLink	
Input:	(A2)	= Address of the module header
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$UnLoad   Unlink module by name.

Assembler Call:	OS9 F$UnLoad	
Input:	D0.W	= Module type/language
	(A0)	= Module name pointer
Output:	(A0)	= Updated past module name
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$Walt   Wait for child process to terminate.

Assembler Call:	OS9 F$Wait	
Input:	None.	
Output:	D0.W	= Deceased child process' process ID
	D1.W	= Child process' exit status code
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.
Possible Errors:	E$NoChld	

OS-9 User Mode System Calls

APPENDIX I: OS-9/68000: I/O SYSTEM CALLS

I$Attach   Attach a new device to the system.

        Assembler Call: OS9 I$Attach

        Input: D0.B   = Access mode (Read_, Write_,Updat_)
                        (A0)   = Device name pointer

        Output: (A2)   = System's device table pointer

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

        Possible Errors: E$DevOvf , E$BMode , E$DevBsy , E$MemFul

I$ChgDir   Change working directory.

        Assembier Call: OS9 I$ChgDir

        Input: D0.B   = Access mode (read/write/exec)
                        (A0)   = Address of the pathlist

        Output: (A0)   = updated past pathname

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code.

        Possible Errors: E$BPNam , E$BMode

I$Close   Close a path to a file/device.

        Assembler Call: OS9 I$Close

        Input: D0.W = Path number

        Output: None.

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code

        Possible Errors: E$BPNum

I$Create   Create a path to a new file.

        Assembler Call: OS9 I$Create

        Input: D0.B   = Access mode (S, I, E, W, R)
                        B1.W   = File attributes (access permission)
                        D2.L   = Initial allocation size (optional)
                        (A0)   = Pathname pointer

        Output:

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code

A-24

OS-9 I/O System Calls

I$Create   Create a path to a new file.

Assembler Call:	OS9 I$Create
Input:	D0.B ▪ Access mode (S, I, E, W, R)
	B1.W ▪ File attributes (access permission)
	D2.L ▪ Initial allocation size (optional)
	(A0) ▪ Pathname pointer
Output:	D0.W ▪ Path number
	(A0) ▪ Updated past the pathlist
Error Output:	cc ▪ Carry bit set.
	D1.W ▪ Appropriate error code
Possible Errors:	E$PthFul, E$BPNam

I$Delete   Delete a file.

Assembler Call:	OS9 I$Delete
Input:	D0.B ▪ Access mode (read/write/exec)
	(A0) ▪ Pathname pointer
Output:	(A0) ▪ Updated past pathlist
Error Output:	cc ▪ Carry bit set.
	D1.W ▪ Appropriate error code
Possible Errors:	E$BPNam

I$Detach   Remove a device from the system.

Assembler Call:	OS9 I$Detach
Input:	(A2) ▪ Address of the device table entry
Output:	None.
Error Output:	cc ▪ Carry bit set.
	D1.W ▪ Appropriate error code
Possible Errors:	E$BPNam

I$Dup   Duplicate a path.

Assembler Call:	OS9 I$Dup
Input:	D0.W ▪ Path number of path to duplicate
Output:	D0.W ▪ New number for the same path
Error Output:	cc ▪ Carry bit set.
	D1.W ▪ Appropriate error code
Possible Errors:	E$PthFul, E$BPNum

A-25

OS-9 I/O System Calls

I$GetStt   Get file/device status.

      Assembler Call: OS9 I$GetStt

      Input: D0.W   - Path number
                             D1.W   - Function code
                             Others - Dependent on function code

      Output: Dependent on function code

      Error Output: cc   - Carry bit set.
                             D1.W   - Appropriate error code

      Possible Errors: E$BPNum

I$MakDir   Make a new directory.

      Assembler Call: OS9 I$MakDir

      Input: D0.B   - Access mode
                             D1.W   - Access permissions
                             D2.L   - Initial allocation size (optional)
                             (A0)   - Pathname pointer

      Output: (A0)   - Updated past pathname

      Error Output: cc   - Carry bit set.
                             D1.W   - Appropriate error code

      Possible Errors: E$BPNam , E$CEF

I$Open   Open a path to a file or device.

      Assembler Call: OS9 I$Open

      Input: D0.B   - Access mode (D, S, E, W, R)
                             (A0)   - Pathname pointer

      Output: D0.W   - Path number
                             (A0)   - (Updated past pathname (trailing spaces skipped)

      Error Output: cc   - Carry bit set.
                             D1.W   - Appropriate error code

      Possible Errors: E$PthFul , E$BPNam , E$Bmode, E$FNA , E$PNNF , E$Share

I$Read   Read data from a file or device.

        Assembler Call: OS9 I$Read

        Input: D0.W   = Path number
                                  D1.L   = Maximum number of bytes to read
                                  (A0)   = Address of input buffer

        Output: D1.L   = Number of bytes actually read

        Error Output: cc   = Carry bit set.
                                    D1.W   = Appropriate error code

        Possible Errors: E$BPNum , E$Read , E$BMode , E$EOF

I$ReadLn   Read a line of text with editing.

        Assembler Call: OS9 I$ReadLn

        Input: D0.W   = Path number
                                    D1.L   = Maximum number of bytes to read
                                  (A0)   = Address of input buffer

        Output: D1.L   = Actual number of bytes written

        Error Output: cc   = Carry bit set.
                                    D1.W   = Appropriate error code

        Possible Errors: E$BPNum , E$Read , E$BMode

I$Seek   Reposition the logical file pointer.

        Assembler Call: OS9 I$Seek

        Input: D0.W   = Path number
                                    D1.L   = New position

        Output: None.

        Error Output: cc   = Carry bit set.
                                    D1.W   = Appropriate error code

        Possible Errors: E$BPNum

I$SetStt   Set file/device status.

        Assembler Call: OS9 I$SetStt

        Input: D0.W = Path number
                              D1.W = Function code
                              Others = Function code dependent

        Output: Function code dependent.

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code

        Possible Errors: E$BPNum

I$Write   Write data to a file or device.

        Assembler Call: OS9 I$Write

        Input: D0.W = Path number
                                D1.L = Maximum number of bytes to write
                              (A0)   = Address of buffer

        Output: D1.L   = Number of bytes actually written

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code

        Possible Errors: E$BPNum , E$BMode , E$Write

I$WritLn   Write a line of text with editing.

        Assembler Call: OS9 I$WritLn

        Input: D0.W = Path number
                                D1.L = Maximum number of bytes to write
                              (A0)   = Address of buffer

        Output: D1.L   = Actual number of bytes written

        Error Output: cc   = Carry bit set.
                              D1.W = Appropriate error code

        Possible Errors: E$BPNum , E$Read , E$BMode

APPENDIX J: OS-9/68000: SYSTEM MODE SYSTEM CALLS

F$AllPD   Allocate process/path descriptor.

Assembler Call:	OS9 F$AllPD	
Input:	(A0)	= Process/path table pointer
Output:	D0.W	= Process/path number
	(A1)	= Pointer to process/path descriptor
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$AllPrc   Allocate process descriptor.

Assembler Call:	OS9 F$AllPrc	
Input:	None.	
Output:	(A2)	= Process descriptor pointer
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.
Possible Errors:	E$PrcFul	

F$AProc   Insert process in active process queue.

Assembler Call:	OS9 F$AProc	
Input:	(A0)	= Address of process descriptor
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code

F$FindPD   Find process/path descriptor.

Assembler Call:	OS9 F$FindPD	
Input:	D0.W	= Process/path number
	(A0)	= Process/path table pointer
Output:	(A1)	= Pointer to process/path descriptor
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code

F$IOQu   Enter I/O queue.

Assembler Call:	OS9 F$IOQu	
Input:	D0.W	= Process number
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$IRQ   Add or remove device from IRQ table.

Assembler Call:	OS9 F$IRQ	
Input:	D0.B	= vector number:
		25-31 for autovectors
		64-255 for vectored IRQ's
	D1.B	= priority (0=polled first, 255=last)
	(A0)	= IRQ service routine entry point (0=delete)
	(A2)	= global static storage pointer (must be unique to device)
	(A3)	= port address
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.
Possible Errors:	E$POLL	

F$Move   Move data (low bound first)

Assembler Call:	OS9 F$Move	
Input:	D0.W	= Source task number (not req'd on Level 1)
	D1.W	= Destination task number (not req'd on Level 1)
	D2.L	= Byte count to copy
	(A0)	= Source pointer
	(A2)	= Destination pointer
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$NProc   Start next process.

Assembler Call:	OS9 F$NProc	
Input:	None.	
Output:	Control does not return to caller.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code

A-30

OS-9 I/O System Mode System Calls

F$RetPD   Return process/path descriptor.

Assembler Call:	OS9 F$RetPD	
Input:	D0.W	= Process/path number
	(A0)	= Process/path table pointer
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$SLink   System link.

Assembler Call:	OS9 F$SLink	
Input:	D0.W	= Desired module type/language (0=any)
	(A0)	= Module name string pointer
Output:	D0.W	= Actual module type/language
	D1.W	= Module attributes/revision
	(A0)	= Updated beyond name string
	(A1)	= Module entry point
	(A2)	= Module pointer
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.
Possible Errors:	E$ModBsy , E$MemFul	

F$SSvc   Service request table initialization.

Assembler Call:	OS9 F$SSvc	
Input:	(A1)	= Pointer to service request initialization table
Output:	None.	
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code.

F$VModul   Verify module.

Assembler Call:	OS9 F$VModul	
Input:	D0.L	= Module group ID
	(A0)	= Address of module
Output:	(A2)	= Directory entry pointer
Error Output:	cc	= Carry bit set.
	D1.W	= Appropriate error code
Possible Errors:	E$KwnMod , E$DirFul , E$BMID , E$BMCRC , D$BMHP	

A-31

OS-9 I/O System Mode System Calls

APPENDIX K: OS-9 Error Codes

Error Number	Error Type	Description	Symbol
000:002	Miscellaneous	Keyboard Quit	
000:003	Miscellaneous	Keyboard Interrupt	
000:064	Miscellaneous	Illegal Function Code	E$IllFnc
000:065	Miscellaneous	Format Error	E$FmtErr
000:066	Miscellaneous	Number Not Found.	E$NotNum
000:067	Miscellaneous	Illegal Argument	E$IllArg
000:102	Uninitialized Trap	Bus Error	E$BusErr
000:103	Uninitialized Trap	Address Error	E$AdrErr
000:104	Uninitialized Trap	Illegal Instruction	E$IllIns
000:105	Uninitialized Trap	Zero Divide	E$ZerDiv
000:106	Uninitialized Trap	Check (CHK)	E$Chk
000:107	Uninitialized Trap	TRAPV	E$TrapV
000:108	Uninitialized Trap	Privilege Violation	E$Violat
000:109	Uninitialized Trap	Trace Error	E$Trace
000:110	Uninitialized Trap	Line 1010 Emulator	E$1010
000:111	Uninitialized Trap	Line 1111 Emulator	E$1111
000:112	Uninitialized Trap	Invalid TRAP #12	E$Resrvd
000:113	Uninitialized Trap	Invalid TRAP #13	E$Resrvd
000:114	Uninitialized Trap	Invalid TRAP #14	E$Resrvd
000:115	Uninitialized Trap	Invalid TRAP #15	E$Resrvd
000:116	Uninitialized Trap	Invalid TRAP #16	E$Resrvd
000:117	Uninitialized Trap	Invalid TRAP #17	E$Resrvd
000:118	Uninitialized Trap	Invalid TRAP #18	E$Resrvd
000:119	Uninitialized Trap	Invalid TRAP #19	E$Resrvd
000:120	Uninitialized Trap	Invalid TRAP #20	E$Resrvd
000:121	Uninitialized Trap	Invalid TRAP #21	E$Resrvd
000:122	Uninitialized Trap	Invalid TRAP #22	E$Resrvd
000:123	Uninitialized Trap	Invalid TRAP #23	E$Resrvd
000:124	Uninitialized Trap	Invalid User TRAP #1	E$Trap
000:125	Uninitialized Trap	Invalid User TRAP #2	E$Trap
000:126	Uninitialized Trap	Invalid User TRAP #3	E$Trap
000:127	Uninitialized Trap	Invalid User TRAP #4	E$Trap
000:128	Uninitialized Trap	Invalid User TRAP #5	E$Trap
000:129	Uninitialized Trap	Invalid User TRAP #6	E$Trap
000:130	Uninitialized Trap	Invalid User TRAP #7	E$Trap
000:131	Uninitialized Trap	Invalid User TRAP #8	E$Trap
000:132	Uninitialized Trap	Invalid User TRAP #9	E$Trap
000:133	Uninit alized Trap	Invalid User TRAP #10	E$Trap
000:134	Uninitialized Trap	Invalid User TRAP #11	E$Trap

Error Number	Error Type	Description	Symbol
000:135	Uninitialized Trap	Uninitialized User TRAP #12	E$Trap
000:136	Uninitialized Trap	Uninitialized User TRAP #13	E$Trap
000:137	Uninitialized Trap	Uninitialized User TRAP #14	E$Trap
000:138	Uninitialized Trap	Uninitialized User TRAP #15	E$Trap
000:139	Miscellaneous	No Permission	E$Permit
000:140	Miscellaneous	Different Arguments	E$Differ
000:141	Miscellaneous	Stack Overflow	E$StkOvf
000:142	Miscellaneous	Illegal Event ID	E$EvntID
000:143	Miscellaneous	Event Name Not Found	E$EvNF
000:145	Miscellaneous	Event Busy	E$EvBusy
000:146	Miscellaneous	Impossible Event Parameter	E$EvParm
000:200	OS-9	Path Table Full	E$PthFul
000:201	OS-9	Illegal Path Number	E$BPNum
000:202	OS-9	Interrupt Polling Table Full	E$Poll
000:203	OS-9	Illegal Mode	E$BMode
000:204	OS-9	Device Table Full	E$DevOvf
000:205	OS-9	Illegal Module Header	E$BMID
000:206	OS-9	Module Directory Full	E$DirFul
000:207	OS-9	Memory Full	E$MemFul
000:208	OS-9	Illegal Service Request	E$UnkSvc
000:209	OS-9	Module Busy	E$ModBsy
000:210	OS-9	Boundary Error	E$BPAddr
000:211	OS-9	End of File	E$EOF
000:212	OS-9	Vector Busy	E$VctBsy
000:213	OS-9	Non-Existing Segment	E$NES
000:214	OS-9	File Not Accessable	E$FNA
000:215	OS-9	Bad Path Name	E$BPNam
000:216	OS-9	Path Name Not Found	E$PNNF
000:217	OS-9	Segment List Full	E$SLF
000:218	OS-9	File Already Exists	E$CEF
000:219	OS-9	Illegal Block Address	E$IBA
000:220	OS-9	Telephone (Modem) Data Carrier Lost	E$HangUp
000:221	OS-9	Module Not Found	E$MNF
000:222	OS-9	No Clock	E$NoClk
000:223	OS-9	Suicide Attempt	E$DelSP
000:224	OS-9	Illegal Process Number	E$IPrcID
000:225	OS-9	Bad Polling Parameter	E$Param
000:226	OS-9	No Children	E$NoChld
000:227	OS-9	Illegal Trap Code	E$ITrap
000:228	OS-9	Process Aborted	E$PrcAbt

A-33

OS-9 Error Codes

Error Number	Error Type	Description	Symbol
000:229	OS-9	Process Table Full	E$PrcFul
000:230	OS-9	Illegal Parameter Area	E$IForkP
000:231	OS-9	Known Module	E$KwnMod
000:232	OS-9	Incorrect Module CRC	E$BMCRC
000:233	OS-9	Signal Error	E$USigP
000:234	OS-9	Non-Existent Module	E$NEMod
000:235	OS-9	Bad Name	E$BNam
000:236	OS-9	Bad Parity	E$BMHP
000:237	OS-9	RAM Full	E$NoRAM
000:238	OS-9	Directory Not Empty	E$DNE
000:239	OS-9	No Task Number Available	E$NoTask
000:240	Device Driver	Illegal Drive Number	E$Unit
000:241	Device Driver	Bad Sector	E$Sect
000:242	Device Driver	Write Protect	E$WP
000:243	Device Driver	CRC Error	E$CRC
000:244	Device Driver	Read Error	E$Read
000:245	Device Driver	Write Error	E$Write
000:246	Device Driver	Not Ready	E$NotRdy
000:247	Device Driver	Seek Error	E$Seek
000:248	Device Driver	Media Full	E$Full
000:249	Device Driver	Wrong Type	E$BTyp
000:250	Device Driver	Device Busy	E$DevBsy
000:251	Device Driver	Disk ID Change	E$DIDC
000:252	Device Driver	Record Is Locked Out	E$Lock
000:253	Device Driver	Non-Sharable File Busy	E$Share
000:254	Device Driver	I/O Deadlock	E$DeadLk
000:255	Device Driver	Device is Format Protected	E$Format

APPENDIX L: ASCII Chart

The 'Char' column contains the ASCII characters. Some of these characters are control characters which are not seen on the keyboard. The 'Hex' column contains the hexadecimal numerical equivalents and the 'Dec' column contains the decimal numerical equivalents.

Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
NUL	00	00	SP	20	32	@	40	64	`	60	96
SOH	01	01	!	21	33	A	41	65	a	61	97
STX	02	02	"	22	34	B	42	66	b	62	98
ETX	03	03	#	23	35	C	43	67	c	63	99
EOT	04	04	$	24	36	D	44	68	d	64	100
ENQ	05	05	%	25	37	E	45	69	e	65	101
ACK	06	06	&	26	38	F	46	70	f	66	102
BEL	07	07	'	27	39	G	47	71	g	67	103
BS	08	08	(	28	40	H	48	72	h	68	104
HT	09	09	)	29	41	I	49	73	i	69	105
LF	0A	10	*	2A	42	J	4A	74	j	6A	106
VT	0B	11	+	2B	43	K	4B	75	k	6B	107
FF	0C	12	,	2C	44	L	4C	76	l	6C	108
CR	0D	13	-	2D	45	M	4D	77	m	6D	109
SM	0E	14	.	2E	46	N	4E	78	n	6E	110
SI	0F	15	/	2F	47	O	4F	79	o	6F	111
DLE	10	16	0	30	48	P	50	80	p	70	112
DC1	11	17	1	31	49	Q	51	81	q	71	113
DC2	12	18	2	32	50	R	52	82	r	72	114
DC3	13	19	3	33	51	S	53	83	s	73	115
DC4	14	20	4	34	52	T	54	84	t	74	116
NAK	15	21	5	35	53	U	55	85	u	75	117
SYN	16	22	6	36	54	V	56	86	v	76	118
ETB	17	23	7	37	55	W	57	87	w	77	119
CAN	18	24	8	38	56	X	58	88	x	78	120
EM	19	25	9	39	57	Y	59	89	y	79	121
SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
FS	1C	28	<	3C	60	\	5C	92	|	7C	124
GS	1D	29	=	3D	61	]	5D	93	}	7D	125
RS	1E	30	>	3E	62	^	5E	94	~	7E	126
US	1F	31	?	3F	63	_	5F	95	DEL (RB)	7F	127

A-35

ASCII Chart

APPENDIX M: FORTH VOCABULARY WORDS

!	;	CONVERT	KEY	SQRT
"	;CODE	COUNT	LAST	STATE
#	<	COUNTER	LEAVE	SWAP
#>	<#	CR	LINK>BODY	TCALL
#OUT	<>	CREATE	LIST	THEN
#S	=	D+	LITERAL	TIB
#TIB	>	D<	LOAD	TIMER
$	>BODY	DEBUG	LOOP	TURNKEY
'	>IN	DECIMAL	L_EXT	TYPE
'=	>R	DEFINITIONS	MACH	U.
(	?DUP	DEPTH	MAKEMODULE	U<
"	?FREE	DISK	MAKECMODULE	UM*
*/	?INCLUDE"	DNEGATE	MATH	UM/MOD
*/MOD	?TERMINAL	DO	MAX	UNTIL
+	@	DOES>	MIN	UPDATE
+!	ABORT	DROP	MOD	VALLOT
+>	ABORT"	DUMP	NEGATE	VARIABLE
+LOOP	ABORT_VECTOR	DUP	NOT	VERBOSE
,	ABS	ELSE	NP	VERIFY
-	AGAIN	EMIT	NUMBER?	VOCABULARY
->	ALLOT	EMPTY	OF	VP
-TRAILING	ALSO	EMPTY-BUFFERS	ONLY	W!
.	AND	END-CODE	OR	W,
."	ASCII	ENDCASE	ORDER	W@
.(	ASSEMBLER	ENDOF	OS-9	WHILE
.R	ASSIGNMODULE	EXECUTE	OVER	WORD
.S	BASE	EXIT	PAD	WORDS
/	BEGIN	EXPECT	PICK	W_EXT
/MOD	BINARY	FILL	QUERY	XOR
0<	BLK	FIND	QUIT	[
0=	BLOCK	FLUSH	R>	[']
0>	BODY>LINK	FORGET	R@	[COMPILE]
1+	BUFFER	FORTH	RECURSIVE	\
1-	BYE	FORTH-83	REPEAT	]
2*	C!	HERE	ROLL	^
2+	C,	HEX	ROT	{
2-	C@	HOLD	SAVE-BUFFERS	
2/	CASE	I	SEAL	
2DROP	CMOVE	I'	SIGN	
2DUP	CMOVE>	IF	SMUDGE	
2OVER	CODE	IMMEDIATE	SPACE	
2SWAP	COMPILE	INCLUDE"	SPACES	
:	CONSTANT	J	SPAN	

Alphabetical Listing of FORTH Vocabulary Words

APPENDIX N: MACH2 EDITION 2 INFORMATION

This appendix documents the differences between Mach2 Edition #1 and
Mach2 Edition #2 for OS-9 version 2.0.

The topics covered are:

New Math Module Support
High Level Driver Creation Supported
Number Base Specification Extended
68020 Long PC-Relative BSR's Supported
Module Name Change
Module Header Information Change

New Math Module Support

The Math1 and Math2 modules found in OS-9 version 1.2 have been combined into one module,
Math, under OS-9 version 2.0. In Mach2 Edition #2, this single math module is supported
via TRAP 15. TRAP 14, which was previously used to access the Math1 module, is now available.

High Level Driver Creation Support Added

Mach2 now contains support for the high level creation of OS-9 device drivers. Any Forth word
which does not reference the Mach2 kernel may be used in the driver. To check for invalid driver
words, store a -1 in VERBOSE and load your driver code. Any invalid kernel references will be
flagged during compilation.

An example Sequential Character File (SCF) device driver will be available in the near future.

MAKEDRIVER

The word MAKEDRIVER is used to turn your loaded code into a proper OS-9 driver module.
MAKEDRIVER is passed the number of bytes of variable space required by the driver and the
start and end address of the driver code in memory. The name for the driver module should follow
MAKEDRIVER:

<#varbytes> <startaddress> <endaddress> MAKEDRIVER <drivemame>

A-37

Mach2 Edition 2 Information

Required driver routines.

Your device driver code must contain the following seven routines:

INIT - Initialize device.
READ - Read a character.
WRITE - Write a character.
GETSTA - Get device status.
SETSTA - Set device status.
TERM - Terminate device.
EXCEPTION - Handles illegal exception.

The functions of these seven routines are described in the OS-9 68000 Operating System Technical Manual. When MAKEDRIVER is executed, it will check for the existence of these seven words. If they are not all present, MAKEDRIVER will present an error message and abort execution.

Driver variables.

To define a driver variable, use the word DRVRVar as follows:

#16 DRVRVar CharWidth

The base address of the driver variable area is kept in the A2 register. The variable storage area for CharWidth would be located 16 bytes into the driver variable area (#16(A2)). Driver variable references are compiled as offsets from the A2 register.

Number Base Specification Extended

By preceeding numbers with the characters, (# $ %) the current BASE of the compiler can be overidden.

DECIMAL   -> #
HEX   -> $
BINARY   -> %

Examples:
$12 = 18 Decimal
%10 = 5 Decimal
#10 = 10 Decimal

68020 Long PC-Relative BSR's Supported

The Mach2 compiler now supports the 32-bit displacement version of the 68020's PC-relative BSR instruction. Execution of the word MC68020 will set the compiler switch which tells the compiler to use 68020 32-bit BSR instructions if necessary. If MC68020 has been executed, and a Forth word references another word which is further than 32K bytes away, the Mach2 compiler will automatically generate the 6 byte 68020 BSR.L instruction ($61FF).

Module Name Change

The main module in the Mach2 file, which used to be called MACH, is now called MACH2.

Module Header Information Fixed

The permission and owner information in the Mach2 module header is now compatible with OS-9 version 2.0.

Appendix N: Mach2 Edition 2 Information

iNDEX

$CLOSE 61,G-211
$CREATE 59,G-211
$DELETE 61,G-211
$OPEN 60,G-211
$WRITE 62,G-211
.ALIGN G-217
?FREE 23
?OS9ERROR 60,64,G-212

A

A6 register 21,69,70,77,80,83,A-3
ALLOT 18,19
ALSO 24-25
assembler 36-44
  comments 39
  data storage allocation 41
  directives 41,G-217,G-218
  examples 15,
  infix 15
  inline math 40
  interactive 36
  local labels 39
  macros 44
  operators 40
  psuedo-instructions 38,39,42
  symbols 41
ASSIGNMODULE 79.
  assembly language definition 81
attributes see 'files'

B

benchmark see 'Sieve'

C

C compiler, OS-9
  example 89-91
  parameter passing conventions 87
  register variables 91
  strings G-3
  see also 'generic format trap module'
case sensitive 5
CODE 37
code space 42,77
  diagram 18
  size of 19
  see also 'HERE' , 'ALLOT'
comments
  in assembler 39
  may be nested G-12
  single line G-207
CONSTANT 41
CREATE 19

D

data area 22,42,66,69,70,77,A-3
data pointer see 'data area'
DC 42,G-217
DCB 43
DEBUG 46
debugger 46-51
  commands 48-51
  display 47
  examples 16
  invoking the debugger 46
  trap module usage 78
  warning 47,G-108
DEFINITIONS 25
device drivers 76,85
dictionary 24
  removing words from 25
dictionary header see 'header'

INDEX

INDEX

H

HEADER G-217,G-218
header 18-20
  diagram A-2
  see also 'names space'
HERE 18,19

I

I$GetStt 56
I$Read 54,55,63
I$ReadLn 54,55
I$SetStt 56
I$Write 54
I$WritLn 54
IMMEDIATE 44
INT 31
interrupt keys 54,55

J

JSR 37,44,51,77,80,G-128,A-6
  see also 'subroutine threading'
jump table 77,G-128
  diagram 21

K

KEY 55

L

LABEL 41,G-218
line editing 54
link field G-76,A-2
LIST (OS-9 command) 62

local variables 12,27-28
  example 27
  operators 28
  register usage A-3
  stack notation 29
loop stack A-4,A-5
  in 'generic' trap module 88

M

'MACH' format trap module 77-82
  assigning trap numbers 78
  calling the trap module 79
  initialization of 80
  reading into memory 81
  selector 77,79
  stack notation 78
  see also 'trap module'
Mach.bit 44
MACH2
  register usage A-3
  Starting up 4
MACHMODULE 77
  example G-178
macro substitution 44,A-7,A-8
MAKEMODULE 84
  example 85
Memory
  availability 12,23
  display 49
  requirements 4
  utility word see '?FREE'
  see also 'code space', 'names space',
    and 'variable space'
mnemonics 45
modules see also 'trap modules'
  disassembler module 45
  executable see 'TURNKEY',
    'MACHMODULE', 'MAKEMODULE'
  math modules 30
MOVEM discussion 80

INDEX

N

named input parameters   see 'local variables'
names space
   diagram 20
   size of 20
   see also 'NP'
NP 20

O

ONLY 24
   example 25
ORDER 26
OS-9/68000 Technical Manuals
   revision 2
OS-9
   error codes A-32-A-34
   error message file 65
   errors 38,64-65
   I/O system calls A-24-A-28
   shell 49,73,77,84
   system mode system calls A-29-A-31
   user mode system calls A-12-A-23
OS9 G-218
   see also 'psuedo-instructions'

P

parameter stack 5,8,29,51
   in 'generic' trap module 88
   register usage A-3
PARAM_PTR 73,G-212
PC-relative 32,44
position-independent 32
precedence (of arithmetic operators) 40
process id 72
process parameter passing 73-75
processes 69,72,73
   child G-9

psuedo-instructions
   DC 42
   DCB 43
   OS9 38,39,44
   TCALL 43

R

Random Block File Manager 54,59
recursion 28
   see also 'local variables'
register usage A-3
relocatable 32
RESPONSE 69,G-212
RTS 37,44,G-208

S

SEAL 26
   diagram 25
search order 24
   diagram 25
   utilities 26
selector see 'trap modules'
self-modifying code 47
Sequential Character File Manager 54
Sieve 15
signals 69-72
   signal intercept routine 69
SMUDGE 44
software exception vector 38,44,45,66,81,G-70
   usage table 78
   see also 'exception handling'
SPAN 55
stack notation 29

INDEX

INDEX