

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

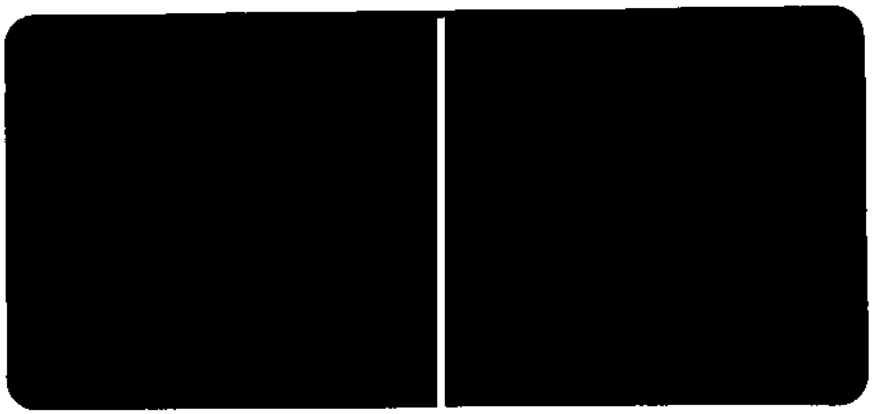
Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Astronics / EADS / Racal 405096* at our website: ***Click HERE***



Racal Instruments

INSTRUCTION MANUAL

7064

DEVELOPMENT MODULE

Publication No. 980699-999

RACAL INSTRUMENTS

Racal Instruments, Inc.
4 Goodyear St., Irvine, CA 92618-2002
Tel: (800) RACAL-ATE, (800) 722-2528, (714) 859-8999; FAX: (714) 859-7139

Racal Instruments, Ltd.
480 Bath Road, Slough, Berkshire, SL1 6BE, United Kingdom
Tel: +44 (0) 1628 604455; FAX: +44 (0) 1628 662017

Racal Systems Electronique S.A.
18 Avenue Dutartre, 78150 LeChesnay, France
Tel: +33 (1) 3955 8888; FAX: +33 (1) 3955 6735

Racal Systems Elettronica Srl.
Strada 2-Palazzo C4, 20090 Milanofiori Assago, Milan, Italy
Tel: +39 (2) 5750 1796; FAX +39 (2) 5750 1828

Racal Elektronik System GmbH.
Frankenforster Strasse 21, 51427 Bergisch-Gladbach 1, Germany
Tel:+49 2204 92220; FAX: +49 2204 21491

Racal Australia Pty. Ltd.
3 Powells Road, Brookvale, NSW 2100, Australia
Tel: +61 (2) 9936 7000, FAX: +61 (2) 9936 7036

Racal Electronics Pte. Ltd.
26 Ayer Rajah Crescent, 04-06/07 Ayer Rajah Industrial Estate, Singapore 0513.
Tel: +65 7792200, FAX: +65 7785400

Racal Instruments, Ltd.
Sun House 13th Floor, 181 Des Voeux Road, Central, Hong Kong
Tel: +852 281 58633, FAX +852 281 58141

RACAL

PUBLICATION DATE: NOVEMBER 1995

Copyright © 1995 by Racal Instruments, Inc. Printed in the United States of America. All rights reserved. This book or parts thereof may not be reproduced in any form without written permission of the publisher.

WARRANTY STATEMENT

Products sold by Racal Instruments, Inc. are warranted to be free from defects in workmanship or materials. Racal Instruments, Inc. will, at its option, either repair or replace any hardware products which prove to be defective during the warranty period. You are a valued customer. Our mission is to make any necessary repairs in a reliable and timely manner.

Duration of Warranty

The warranty period for this Racal Instruments, Inc. hardware is three years, except software and firmware products designed for use with Racal Instruments, Inc. Hardware are warranted not to fail to execute their programming instructions due to defect in materials or workmanship for a period of ninety (90) days from the date of delivery to the initial end user.

Return of Product

Authorization is required from Racal Instruments before you send us your product for service or calibration. Call your nearest Racal Instruments support facility. A list is located on the last page of this manual. If you are unsure where to call, contact Racal Instruments, Inc. Customer Support Department in Irvine, California, USA at 1-800-722-3262 or 1-949-859-8999 or via fax at 1-949-859-7309. We can be reached at: helpdesk@racalate.com.

Limitation of Warranty

Racal Instruments, Inc. shall be released from all obligations under this warranty in the event repairs or modifications are made by persons other than authorized Racal Instruments service personnel or without the written consent of Racal Instruments.

Racal Instruments, Inc. expressly disclaims any liability to its customers, dealers and representatives and to users of its product, and to any other person or persons, for special or consequential damages of any kind and from any cause whatsoever arising out of or in any way connected with the manufacture, sale, handling, repair, maintenance, replacement or use of said products.

Representations and warranties made by any person including dealers and representatives of Racal Instruments, Inc., which are inconsistent or in conflict with the terms of this warranty (including but not limited to the limitations of the liability of Racal Instruments, Inc. as set forth above), shall not be binding upon Racal Instruments, Inc. unless reduced to writing and approved by an officer of Racal Instruments, Inc.

Except as stated above, Racal Instruments, Inc. makes no warranty, express or implied (either in fact or by operation of law), statutory or otherwise; and except to the extent stated above, Racal Instruments, Inc. shall have no liability under any warranty, express or implied (either in fact or by operation of law), statutory or otherwise.

PROPRIETARY NOTICE

This document and the technical data herein disclosed, are proprietary to Racal Instruments, and shall not, without express written permission of Racal Instruments, be used, in whole or in part to solicit quotations from a competitive source or used for manufacture by anyone other than Racal Instruments. The information herein has been developed at private expense, and may only be used for operation and maintenance reference purposes or for purposes of engineering evaluation and incorporation into technical specifications and other documents which specify procurement of products from Racal Instruments.

FOR YOUR SAFETY

Before undertaking any troubleshooting, maintenance or exploratory procedure, read carefully the **WARNING** and **CAUTION** notices.

This equipment contains voltage hazardous to human life and safety, and is capable of inflicting personal injury.

If this instrument is to be powered from the AC line (mains) through an autotransformer, ensure the common connector is connected to the neutral (earthed pole) of the power supply.

Before operating the unit, ensure the conductor (green wire) is connected to the ground (earth) conductor of the power outlet. Do not use a two-conductor extension cord or a three-prong/two-prong adaptor. This will defeat the protective feature of the third conductor in the power cord.

Maintenance and calibration procedures sometimes call for operation of the unit with power applied and protective covers removed. Read the procedures and heed warnings to avoid "live" circuit points.

Before operating this instrument:

1. Ensure the instrument is configured to operate on the voltage at the power source. See Installation Section.

2. Ensure the proper fuse is in place for the power source to operate.

3. Ensure all other devices connected to or in proximity to this instrument are properly grounded or connected to the protective third-wire earth ground.

If the instrument:

- fails to operate satisfactorily
- shows visible damage
- has been stored under unfavorable conditions
- has sustained stress

Do not operate until performance is checked by qualified personnel.

Racal Instruments

EC Declaration of Conformity

We

Racal Instruments Inc.
4 Goodyear Street
Irvine, CA 92718

declare under sole responsibility that the

7064+	**Single Slot Prototyping Module w/Interface**	**RD70-0010-000**
7064-1+	**Double Slot Prototyping Module w/Interface**	**RD70-0010-001**
7064-1P+	**Double Slot Prototyping Module w/Interface (GA Hole Pattern)**	**RD70-0034-002**
7064-2+	**Single Slot Prototyping Module w/12.5 MHz Interface**	**RD70-0010-002**
7064-6+	**Single Slot Prototyping Module**	**RD70-0011-000**
7064-7+	**Double Slot Prototyping Module**	**RD70-0011-001**
7064-8+	**Double Slot Prototyping Module w/12.5 MHz Interface**	**RD70-0010-003**

conforms to the following Product Specifications:

EMC: CISPR 11:1990/EN 55011 (1991): Group 1 Class A
IEC 801-2:1991/EN 50082-1 (1992): 4 kV CD, 8 kV AD
IEC 801-3:1984/EN 50082-1 (1992): 3 V/m, 27-500 MHz
IEC 801-4:1988/EN 50082-1 (1992): 1 kV

Supplementary Information:

The above specifications are met when the product is installed in a Racal Instruments certified mainframe with faceplates installed over all unused slots, as applicable.

The product herewith complies with the requirements of the EMC Directive 89/336/EEC.

Irvine, CA, November 12, 1996

Quality Manager

Amendment to Racal Instruments

Model 7064 Instruction Manual

March 1996

Section 2 was replaced. Additional information was added to Paragraph 2.5.

TABLE OF CONTENTS

SECTION 1 GENERAL DESCRIPTION

1.1 Introduction 1-1
1.2 General Description 1-1
1.3 7064 Specifications 1-2
1.4 Options 1-3

SECTION 2 INSTALLATION

2.1 Introduction 2-1
2.2 Unpacking and Inspection 2-1
2.3 Address Switch Settings For Configuration Control 2-1
2.4 VXIbus Interrupt Handler Setting 2-2
2.5 7064 To VXIvus Mainframe Installation 2-2
2.6 Power-Up Self-Test Initialization 2-4
2.6.1 Self-Test 2-4
2.6.2 Board ID Check 2-5
2.6.3 Port Initialization 2-5
2.6.4 VXIbus Interface Initialization 2-5
2.6.5 Debugger Initialization 2-5
2.6.6 Invoke User Routine 2-5
2.7 Local Bus Usage With 7064 Breadboard 2-6
2.8 Installing The VSD-1 Serial Debugger 2-6

SECTION 3 BREADBOARD OPERATION

3.1 Introduction 3-1
3.2 Command Parsing 3-1
3.3 Input/Output Port Configuration 3-1
3.3.1 Output Port Reset Polarity 3-1
3.3.2 Port Configurations 3-2
3.3.3 Port Control 3-2
3.4 Breadboard Port Addressing Schemes 3-4
3.5 Base Numbers 3-4
3.6 Local Command Set 3-5
3.6.1 Commands 3-5

SECTION 4 68000 INTERFACE

4.1 Introduction 4-1
4.2 The Message-Based Interface Software Architecture 4-1
4.2.1 Overview 4-1
4.2.2 Calling Conventions 4-2
4.2.3 VXIbus Interface 4-2
4.2.4 Timer 4-2
4.3 Memory Space 4-3

TABLE OF CONTENTS (continued)

4.3.1	EPROM Organization	4-3
4.3.2	RAM	4-4
4.3.3	EEPROM (Non-Vol)	4-6
4.4	Peeking and Poking The 68000	4-12
4.5	Users Application Specific Code	4-15
4.5.1	Loading Application Code	4-15
4.5.2	S-Record Download	4-15
4.5.3	Executing Application Code	4-18
4.6	Application Specific Command Parser	4-19
4.6.1	Selecting a Block of Code As Application Command Parser	4-20
4.6.2	Control Flow Models	4-20
4.6.3	Character I/O Available to Application Command Parser	4-20
4.7	Accessing Message-Based Interface System Services	4-22
4.7.1	Compiling and Linking For Message-Based Interface System Services	4-23
4.7.2	Linking - How The Jump Table Works	4-27
4.8	Putting It All Together	4-27

SECTION 5 REGISTER-BASED BREADBOARD APPLICATIONS

5.1	Introduction	5-1
5.2	Registers	5-1

SECTION 6 MAINTENANCE

6.1	Test Procedure	6-1
6.2	Required Equipment	6-1
6.3	Tests	6-1
6.3.1	Visual Tests	6-1
6.3.2	Functional Tests	6-1
6.4	7064 Test Software	6-3

SECTION 1 GENERAL DESCRIPTION

1.1 INTRODUCTION

This manual contains information on how to install and operate the 7064 in a VXIbus environment. It describes the function of the 7064 under control by the 96 buffered I/O lines, or via the 68000 bus lines.

1.2 GENERAL DESCRIPTION

The 7064 is a message-based development module made up of a removable message-based interface, and a breadboard card consisting of seventy square inches of breadboard area for prototyping and development of digital and analog circuits.

The user circuitry developed on the breadboard can be controlled either by the twelve 8-bit individually configurable ports, or via the 68000's bus directly. The upper half of the 68000 address space is available on the Message-Based Interface module for user development.

The breadboard area is the front two-thirds of the module, and consists of two separate arrays with holes on .1 inch centers. The arrays are divided into group A and group B areas. Each area is subdivided into lettered columns and numbered rows for user convenience. There are two uncommitted power distribution grids in each array available to the user. These grid planes may be connected to various power supplies provided by the VXI backplane. These two layers are called "Top" and "Bot" (Bottom). There are also two power planes dedicated to +5V and ground.

One Message-Based Interface module can control up to twelve breadboard cards via the local bus (see VXIbus Revision 1.3 specs for definition), allowing one to design multiple module VXIbus instruments using a single interface board. The Message-Based Interface module must be installed on the left-most breadboard card.

1.3 7064 SPECIFICATIONS

VXIbus Revision 1.3 compatible	IEEE 488.2 Compatible

CPU :	68000
CPU Clock Rate :	8 MHz (12.5 MHz option available)
Non-Volatile Memory :	16K expandable to 64K
RAM Memory :	64K expandable to 256K
ROM Memory :	128K expandable to 512K
Buffered I/O :	96 Channels
Breadboard Area :	70 Square inches

VXIbus Interface

Manufacturer ID no.:	3915 (user programmable)
Model no.:	3915 (user programmable)
Logical Addressing:	Dynamic or Static configured (1 to 254)
Address Space :	A16 (A24 on 7064-10)
Device Class :	Message Based Slave Device
Interrupt Levels :	Programmable
Trigger Levels :	Programmable TTL levels
Shared Memory :	Available for 7064-10

Word Serial Protocol Supported:

Byte Available	Unsupported Commands
Byte Request	Trigger
Write to logical address register	Clear
Read Status Byte	Abort Normal Operations
Read Protocol Error	Assign Interrupter Line
Asynchronous Mode Control	Begin Normal Operations
Begin Normal Operations Top	Control Event
Control Response	End Normal Operations
Read Interrupter Line	Read Protocol

1.4 OPTIONS

PART NO.	MODEL/OPTION	DESCRIPTION
70-0010-000	7064	Single slot prototyping card w/8MHz interface
70-0018-000	7064-10	Prototyping card with dual ported ram and DMA
70-0010-001	7064-1	Double slot prototyping card w/8MHz interface
70-0010-002	7064-2	Single slot prototyping card w/12.5 MHz intfc
70-0010-003	7064-8	Double slot prototyping card w/12.5 MHz intfc
70-0011-000	7064-6	Single slot prototyping card w/No interface
70-0011-001	7064-7	Double slot prototyping card w/No interface
70-0014-000	Option 05	8MHz 68000 message based interface
70-0014-001	Option 14	12.5MHz 68000 message based interface
70-0017-001	Option 11	Single Wide VXIbus enclosure Kit
70-0017-002	Option 12	Double Wide VXIbus enclosure Kit
70-0017-003	Option 13	Triple Wide VXIbus enclosure Kit
70-0019-001	Option 08	256K RAM upgrade for interface (factory instld)
70-0019-002	Option 09	256K ROM upgrade for interface
70-0015-000		Serial debugger (needed for firmware dvpt)
70-0011-000	Option 02	68000 C compiler, assembler and linker
70-0020-000	Option 04	Windowing debugger kit, incls 70-0011/0015-000

SECTION 2 INSTALLATION

2.1 INTRODUCTION

This section describes the unpacking, inspection, set-up and installation of the Model 7064 development module in a C-size mainframe.

2.2 UNPACKING AND INSPECTION

The following parts constitute the 7064 Module P/N 70-0010-000:

Single-Slot Breadboard Module	P/N 70-0011-000
Message-Based Daughter Board	P/N 70-0014-000
7064 User Manual	

1. Before unpacking the 7064 module, check the exterior of the shipping carton for any damage. If the shipping carton is damaged, inform the carrier immediately.

2. Remove the 7064 module and inspect it for damage. If any damage is apparent, inform the carrier immediately. Retain shipping carton and packing material for the carrier's inspection.

3. Verify that the pieces contained in the package you received match the parts list shown above. Notify Racal Instruments if the module appears damaged in any way. Do not attempt to install a damaged module into a VXI chassis.

4. The 7064 module is shipped in an anti-static bag to prevent electrostatic damage to the module. Do not remove the module from the anti-static bag unless it is in a static-controlled area.

2.3 ADDRESS SWITCH SETTINGS FOR CONFIGURATION CONTROL

The 7064 has an internal 8-position address switch used to determine dynamic configuration or static configuration. It is located on the top of the module accessible through the case. Any setting other than 255 indicates static configuration.

Dynamic configuration is an optional alternative method of assigning logical address to VXIbus devices, and is defined in detail in Section F of the VXIbus System Specification Revision 1.4. In the ON position, the switch is set to logical 1, and in the OFF position to a logical 0.

The user can select any logical address from 1 to 254 for static configuration.

NOTE

Logical address 0 is not allowed.

For dynamic configuration, set the address switch to 255.

NOTE

The 7064 Breadboard Module is shipped with the address set for dynamic configuration. Refer to the Resource Manager's manual for details about its addressing methods.

2.4 VXIBUS INTERRUPT HANDLER SETTING

One programmable interrupt line is provided on the 7064 module. This line is assigned by using the *assign interrupter line* word serial protocol command (See Page 182 of Section E, Revision 1.4 of the VXIbus Specifications). The *Int_ID* is set to 1.

2.5 7064 TO VXIBUS MAINFRAME INSTALLATION

The 7064 module is ready for operation when shipped. The address switch is set to 255 to allow for dynamic configuration by the VXI Slot 0 Resource Manager. See Section 2.3 if static configuration (any switch setting other than 255) is preferred.

To install the 7064 in a C-size VXI chassis, use the following instructions:

1. Ensure power is OFF.

2. Configure the interrupt daisy chain on the backplane to bypass empty slots, per VXIbus specifications.

3. Remove the front cover of the VXI chassis, and slide the 7064 into the appropriate slot with the LED's towards the top (or to the left when using a horizontal chassis).

To ensure reliable VXIbus communications when using 7064 modules without a VXIbus communications interface (i.e., 7064-6, 7064-7, and 7064-25 series), the user must note the following. The VME Specification requires that the Interrupt Acknowledge (IACK) and BUSGRANT (BG) signal lines be daisy-chained from slot-to-slot across the backplane. This means that for each slot the BUSGRANT and IACK signals are input on Row A of the P1

connector and output on Row C of the P1 connector. When there is no VXIbus interface installed in the 7064 module, the user must ensure that these signals are passed onto the next slot. There are three ways this can be done, depending on the backplane installed in your VXIbus chassis. You will need to consult the chassis manual to verify the type of backplane you have.

1. **VXIbus Backplanes with Bus Grant and Interrupt Acknowledge DIP Switches.** - The user must close the switches in the slot where the 7064 will be installed.

2. **VXIbus Backplanes with Active Automatic Daisy-Chain**. - The Bus Grant and Interrupt Acknowledge signals will be automatically passed to the next slot. The user does nothing.

3. **VXIbus Backplanes with Auto-Configuration P1 Connectors**. - With this style of connector, the BUSGRANT and IACK daisy-chains are broken as soon as the module is installed in a slot. The user must reconnect the daisy-chain in the 7064 module. This is done by shorting the pins on the P3 or J101 connector as shown in Table 2.1 below.

Table 2.1 Signal Shorts For Chassis With Auto-Configuration P1 Connectors

SIGNAL	P3 or P101 SHORT
BG0IN* and BG0OUT*	Pin 4 Row B to Pin 5 Row B
BG1IN* and BG1OUT*	Pin 6 Row B to Pin 7 Row B
BG2IN* and BG2OUT*	Pin 8 Row B to Pin 9 Row B
BG3IN* and BG3OUT*	Pin 10 Row B to Pin 11 Row B
IACKIN* and IACKOUT*	Pin 21 Row A to Pin 22 Row A

Bus Granp (Part Number 401951-003) has been designed to perform this task and may be ordered from the Racal Parts Department. The jumper should be located in P3 or J101, pins 1 through 22 as shown in Figure 2.1.

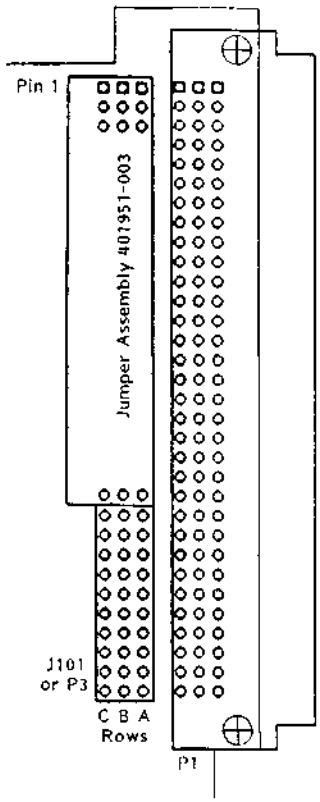


Figure 2.1 Installation of Bus Grant Jumper

2.6 POWER-UP SELF-TEST INITIALIZATION

Before turning on the VXIbus mainframe, make sure a Slot 0 with a Resource Manager is present. Upon power-up of the system, the 7064 goes through the following power-up sequence:

- The 7064 breadboard module has two LED's on the front panel - FAIL and ACCESS. The LED functions are:

FAIL	On during a self-test Off when the self-test has successfully completed
ACCESS	Blinks on when the VXIbus is accessing the module's logical address

At power-up, the 7064 goes through a series of operations to ensure proper initialization and establish the proper start-up state. A brief description of this power-up sequence follows.

2.6.1 Self-Test

Tests are performed on ROM, RAM, Non-Vol and the timer to ensure they are operating correctly.

2.6.2 Board ID Check

Each breadboard may have one or more board ID bytes selected by the user. The user selects a port, a value and a mask if necessary. This information is stored into non-vol by the user and verified during the power-up sequence. This check is bypassed if not specified by the user.

2.6.3 Port Initialization

All I/O ports configured as outputs are initialized to either all 0's or all 1's during system reset (jumper selectable on each breadboard). The user can also specify initial values for ports and have them stored in Non-vol. If any of these values have been specified, they are set at this point in the power-up sequence. The operation of the board ID is described in more detail in Section 3.2.1

2.6.4 VXIbus Interface Initialization

At times, it may be desirable for the user to specify an initial value for one or more of the VXIbus registers. For example, the user specifies a model code. This requires the new model code to be placed in the "device-type" VXIbus register. The user can specify these values and have them stored in Non-Vol. If any of these values have been specified, they are set up at this point in the power-up sequence.

2.6.5 Debugger Initialization

If the debugger ROM is present, it requires initialization to be performed at this point.

2.6.6 Invoke User Routine

The user may want to gain control after the 7064 has been initialized, but before the main loop starts running. The user can specify a vector for this power-up routine. This vector is stored in non-vol. If the vector has been specified, the user's initialization routine is called at this point.

NOTE

At this point, invoke the main routine if the main processing loop is not desired.

At the conclusion of the self-test (less than 4.9 seconds), the FAIL LED should be extinguished. An illuminated FAIL LED is an indication of a module failure. If this occurs, the module should be removed from the VXIbus chassis, and examined for damage or improper installation.

2.7 LOCAL BUS USAGE WITH 7064 BREADBOARD

The VXIbus has a provision for interconnecting adjacent cards through a local bus. The bus consists of 12 lines which jumper card together. A card on the left of another will have connections on the "C" side of P2 tied to the "A" side of P2 for the card to its right. (Refer to the VXIbus specification for exact pinouts.)

2.8 INSTALLING THE VSD-1 SERIAL DEBUGGER

1. Remove the thirteen #4 x 1/4 inch and the three #2 x 3/16 inch screws retaining the top cover of your breadboard module. Remove the cover and set it aside.

2. Remove the EPROM located at U1 of the VXIbus interface board using a PLCC extraction tool (i.e., Burndy P/N 855045). This EPROM must be removed in order for the breadboard to function with a debugger installed.

3. Remove the two #4 x 5/8 inch screws located between P100 and P102 of the VXI interface board along with their two flat and two split lock washers. Retain this hardware for use when the serial debugger assembly is removed.

4. The serial debugger assembly is installed directly above the VXIbus interface board. The two mounting spacers on the debugger assembly should be aligned to the two mounting points where the screws that were removed came from. Use the two #4 x 1 1/8 inch screws along with the flat and split lock washers provided with the serial debugger to hold the debugger in place.

5. If the debugger assembly is correctly situated, its P1 connector will align directly above the VXIbus interface's P100 connector. Install the short ribbon cable assembly between these two connectors, taking care to align them correctly.

CAUTION

Incorrect installation of this cable can result in damage to the debugger and/or VXIbus interface assemblies.

6. Locate the 9-foot cable and D-SUB connector supplied with the debugger. Install the phone jack connector on the cable to the D-SUB connector and the other end to P2 of the serial interface board. Note that P2 is keyed.

The installation of the serial debugger is now complete. The cover of the breadboard cannot be reinstalled while the serial debugger is in place. If the breadboard is installed in a chassis, the slot to its right must be left open.

SECTION 3 BREADBOARD OPERATION

3.1 INTRODUCTION

There are two methods of controlling user-defined circuitry on the breadboard area. The first way is to use the upper half of the 68000 address space and the appropriate software tools. The second and easier approach is to use the 96 buffered I/O lines available on the breadboard card. This section discusses the latter approach.

For more complex programming, the user may download user-defined code which can run interactively with the local command set. This is described in more detail in Section 4.

3.2 Command Parsing

User-defined commands can easily be added and used in conjunction with the local command set of the breadboard. When a command is sent to the 7064 using word serial protocol, it is received and parsed by the command parser. If the command matches a command from the local command set, the appropriate routine is called. If no matching command is found (user-defined command), the command is placed in the secondary input queue. The user routine called from the main loop will then retrieve the characters from the secondary input queue and parse for their own commands. The nvsetvect command can be used to allow the 7064 parser to be bypassed, allowing the users parser to take control of the 7064. If there are no main loop user routines, the main loop generates an error for the characters placed in the secondary queue. This and other complex tasks are described in more detail in Section 4.

3.3 INPUT/OUTPUT PORT CONFIGURATION

A set of local commands are available to the user to allow for the control of the user-defined circuit through the twelve I/O ports available. This section describes the control of these lines.

The 7064 breadboard has twelve 8-bit ports. Each Port can be individually configured for input or output. Three lines are supplied for the user to control the type of port and its operation. The port outputs are designed to accept a 14-pin dual-in-line connector for the user to easily wire into the breadboard area.

3.3.1 Output Port Reset Polarity

All outputs are set to a known level during reset. The user can select whether all the outputs will be set to a logic 1, or all outputs will be set to a logic 0 by simply doing the following:

following:

- To cause all outputs to be set to a logic 1 during reset, connect pin 1 to pin 2 on JP1 of the breadboard card (See Drawing No. 52-0010-000 for location of JP1).

- To cause all outputs to be set to a logic 0 during reset, connect pin 2 to pin 3 on JP1.

NOTE

The 7064 breadboard module is shipped with no connections on JP1. The user must configure JP1 to achieve consistent output port reset levels. JP1 has no effect on ports configured for input.

3.3.2 Port Configurations

The 7064 ports can be configured in three ways:

- Clocked Inputs. The user supplies a clock signal that stores data in the port. At a later time, the port can be read from the VXIbus.

- Buffered Input. Data on the input lines are read when the port is read from the VXIbus. No clock signal is required.

- Latched Output with Readback. Data written to an output port through the VXIbus is held on the data lines until another write changes the data.

3.3.3 Port Control

There are two lines that control the configuration of a port, O/Ix! and LAT/BUFFx!, where x is the associated port number (0 to 11):

- The O/Ix! line: Used to configure the port as an input or an output. When the O/Ix! line is held low, the port functions as an input. When the O/Ix! line is held high, the port functions as an output.

- LAT/BUFFx! line: Used to configure an input port as either a clocked input or a buffered input. When the LAT/Buffx! line is held low, the input port functions as a buffered input. When the LAT/Buffx! line is held high, the input port functions as an clocked input. If the port is configured as an output, this

line should be held low (configured as a buffer), then the

data latched on the output can be read back with a read from the VXIbus. This allows bit operations which modify a single bit, leaving the other bits in a port unchanged.

Table 3-1 correlates the two control lines to their operation:

Table 3-1

O/Ix!	LAT/BUFFx!	OPERATION
0	0	Buffered Input
0	1	Clocked Input
1	0	Output With Read Back
1	1	Not Recommended

NOTE

These two control lines do not have to be static. They can be connected to data pins on an output port, or they can be connected to some user control lines. This allows ports to be used for bidirectional communication, or other more complex functions.

There is one line that is used as a clock line, where x is the associated port number (0 to 11):

- The CLKINx line: Used to clock data into the port when the port is configured as a clocked input. A rising edge on this line clocks data into the port. Once clocked in, the data can be read over the VXIbus. If the port is configured as a buffered input, this line has no affect on the port operation.

NOTE

The 7064 breadboard is shipped with no jumpers on the port control lines. This makes the default port configuration a buffered input. Drawing number 50-0010-000, Sheets 6 to 11, show the pin numbers for each one of the three configuration lines per port.

3.4 BREADBOARD PORT ADDRESSING SCHEMES

There are two groups of commands:

- port numbers
- card ports

The software commands in the local command set that do not begin with a 'c' use straight port numbers. Port numbers run from 0 to 2^{15} - 1 (32767). Port space is divided into 16 cards. Each card potentially has 2048 ports. This means a card with its address set to 0 has ports 0 to 2047. And, with its address set to 1, it has ports 2048 to 4095, etc. Hardware is only provided for the lower twelve ports on each card. The user may provide hardware for additional ports.

Another group of commands is available to perform the same functions, but accounts for the offset of each group of ports on a card. These are the commands that start with a 'c'. This may be more convenient for a system with a large number of breadboard modules. These commands use a 'card' (0 to 15) and a 'cport' (0 to 2047) to locate the actual address.

When using one breadboard only, the card address can be set to 0, and the first set of commands can be used. This has the advantage of shorter commands.

Example:

The command "setbit(4096,0);" and "csetbit(2,0,0);" both set port 0, bit 0 on card number 2 to 1.

3.5 BASE NUMBERS

Numbers used for cards, ports, values, etc. can be represented in any of the standard bases. The default base is decimal. Hex numbers are represented by a leading '#H' in the number. For example, 255 is represented by #HFF. Upper case and lower case are the same and can be mixed freely. Binary numbers are represented by a leading '#B'. For example, 255 in binary is #B11111111. Octal numbers are represented by a leading '#Q'. For example, 255 in octal is #Q377. All base indicators can be either upper or lower case. C-style hex and octal numbers are also accepted.

Commands that return values can have the values return in binary, octal, decimal or hex. The format of the returned value is determined by the base command. This only affects returned values.

3.6 LOCAL COMMAND SET

3.6.1 Commands

The syntactic style of the 7064 command set is reminiscent of the C-language function calls. All built-in and user-supplied commands are invoked by name, as if calling C functions. This section lists the command set required for control of the breadboard. For an in-depth discussion on the complete command set syntax, see Appendix B.

3.6.1.1 Port Commands

There are twelve 8-bit ports on a standard breadboard. The port commands are for reading and writing to these ports.

3.6.1.2 Output Format Command

base(base_value);

Used to set the output number format for return values:

Valid values of base_value are:

1	Binary
8	Octal
10	Decimal
16	Hex

The default value is decimal. When a base_value other than decimal is used, the numbers are formatted as described above.

Example:

1. The number 255 could be returned in any of these formats:

255	Decimal
#HFF	Hex
#Q377	Octal
#B11111111	Binary

2. The command "base(1);" sets the base to binary.

3.6.1.3 Bit Commands

```
setbit(port,bit_number);
rstbit(port,bit_number);
outbit(port,bit_number,value)
pulsehi(port,bit_number);
pulselo(port,bit_number);
inbit(port,bit_number);
csetbit(card,cport,bit_number);
crstbit(card,cport,bit_number);
coutbit(card,cport,bit_number,value);
cpulsehi(card,cport,bit_number);
cpulselo(card,cport,bit_number);
cinbit(card,cport,bit_number);
```

These commands are provided for bit level manipulation. These commands use a bit_number to identify which bit is to be operated on. Bit_numbers range from 0 to 7. 0 is the least significant bit while 7 is the most significant bit.

In order to modify bits in a port without changing others, it is necessary to have the ports configured so the current state can be read. This is done on an output port by leaving the port configured for readback. For this operation, the LAT/BUFFx! is held low (See Section 3.1 for explanation).

setbit and **csetbit** - bit is set to 1
rstbit and **crstbit** - bit is set to 0
outbit and **coutbit** - bit is set to value
pulsehi and **cpulsehi** - the bit is set to 1 and then to 0
pulselo and **cpulselo** - the bit is set to 0 and then to 1
inbit and **cinbit** - reads and returns the value of the bit i.e. 0 or 1

3.6.1.4 Byte Commands

```
outportb(port,byte);
inportb(port);
coutportb(card,cport,byte);
cinportb(card,cport);
```

These commands are provided for operations on bytes of data.

outportb and **coutportb** - port is set to the value
inportb and **cinportb** - reads and returns the value read from the port

Example:

"outportb(2,10);" followed by "inportb(2)" writes to port 2 with a hex decimal value of 10, and reads this value back.

3.6.1.5 Word Commands - Two Consecutive Ports

```
outport(port,word);
inport(port);
coutport(card,cport,word);
cinport(card,cport);
```

Commands are provided for operations on 16 bit words. In this case, two consecutive ports are used as a 16-bit word. The port with the lower address is the least significant byte, and the port with the higher address is the most significant byte. The port specified in these commands is the lower address, which may be even or odd. The lower address port will be written first.

outport and **coutport** - ports are set to 16 bit value
inport and **cinport** - return the 16 bit value read from the ports.

Example:

"coutport(0,0,#B0000000011111111);" followed by "cinport(0,0);" writes 255 in binary to ports 0,1 of card 0, and reads back the port value in decimal.

3.6.1.6 7064 Trigger Commands

```
triggin(trig_line);
trigout(trig_line);
triggen( );
```

These commands allow the 7064 to utilize any one of the eight(8) VXIbus TTL trigger lines.

triggin - defines which TTL trigger line the 7064 will respond to.
trigout - defines which TTL trigger line the 7064 will generate a trigger on.
triggen - generates a trigger by the 7064 on the defined "trigout" line.

Example:

"triggin(1);" followed by "trigout(1);" sets up the 7064 to respond to and generate triggers on VXIbus TTL trigger line 1. Every time the command "triggen(1);" is sent, the 7064 triggers itself.

NOTE

"triggin(8)" disables triggin. The default value is set to 8.

SECTION 4 68000 INTERFACE

4.1 INTRODUCTION

The Message-Based Interface has a simple and flexible software architecture that allows users to load and execute their own application code in a variety of ways. Additionally, the Message Based Interface provides several system services at the level of C-language function calls. Many Message-Based Interface system configuration parameters are programmable giving the user the ability to custom-tailor the Message-Based Interface for a particular application.

Users may load many individually named blocks of application-specific code, and later invoke selected blocks by name. The user may also load an application-specific command parser which may also invoke these blocks, and Message-Based Interface system services.

It is advantageous to users to load and execute their own code for speed-critical applications. By doing so, the user's application can execute at full speed of the Message-Based Interface's on-board Motorola 68000 microprocessor.

4.2 THE MESSAGE-BASED INTERFACE SOFTWARE ARCHITECTURE

4.2.1 Overview

The Message-Based Interface has a single-threaded control-flow architecture. All Message-Based Interface firmware runs in the supervisor mode, and utilizes the supervisor stack pointer. After the initial power-up sequence, the Message-Based Interface command parser gains control. It loops endlessly, examining the character input stream for Message-Based Interface commands.

Named blocks of executable application code may be loaded into a reserved memory area of the Message-Based Interface address space. At run time, these blocks may be invoked by name. Syntactically, these invocations look like C-language function calls. Also, an application-specific command parser may be loaded and invoked in two ways. First, whenever the Message-Based Interface is waiting for input, it will make a function call to the application parser rather than "busy wait" on input. When the machine is idle, the application parser is repeatedly called. Second, the user may explicitly command the Message-Based Interface to redirect the subsequent input stream to the application-specific parser (the exact details of this maneuver are described in the section on Message-Based Interface command syntax).

A jump table, located just above the 68000 exception vectors, is maintained in EPROM. This jump table provides linkage between application specific code and Message-Based Interface system services.

ASCII command strings arriving over the VXI interface are placed into a primary input queue. Each position in the queue holds 16-bits of data. VXI status information (like "EOI asserted") is encoded into the upper 8 bits of each piece of data. These 16-bit values are referred to as "raw data", or "raw characters". Application specific command strings are placed into a secondary queue. All VXI status information in the upper 8 bits is preserved. A suite of functions is available to retrieve raw and stripped characters from the input queues, and to query the VXI status bits.

A heap is maintained, and its overall size is programmable. The default size is 4Kb. The Message-Based Interface provides memory management functions that allocate and reclaim blocks of storage from this heap.

The Message-Based Interface maintains a global symbol table. At run time, it is used to bind identifiers (i.e. ASCII strings) to the absolute addresses of blocks of application specific code.

4.2.2 Calling Conventions

All Message-Based Interface system services obey C-language calling conventions as specified by the Intermetrics C Cross Compiler. See the Cross Compiler's Manual for complete details.

4.2.3 VXIbus Interface

The VXI interface is implemented with dual-port RAMs. The RAMs and interface I/O support circuitry are located in this area.

User space is the upper half of memory. The user may use this space in conjunction with any user supplied code and circuitry to implement special functions and handle special needs.

All of the I/O ports are located in the local bus space. Reads and writes to this area cause transfers across the local bus. These cause reads or writes to the I/O ports.

4.2.4 Timer

A timer is used to maintain a real time clock with the time since power-up initialization. A set of routines are provided to allow the user access to the timer.

4.3 MEMORY SPACE

The memory space of the 68000 is divided into two major areas. The upper half of memory is available for the user. This 8 Mbyte space has no breadboard circuitry or memory located in it. The lower half of memory contains all of the breadboard's I/O ports, the onboard memory and circuitry.

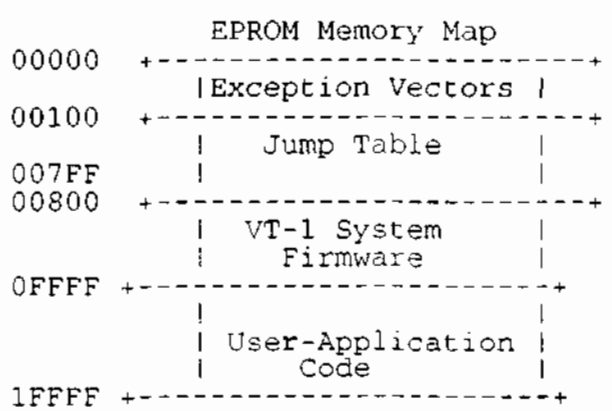
The address space is decoded as follows:

000000 - 1FFFFF	ROM
200000 - 3FFFFF	Reserved
400000 - 5FFFFF	RAM
600000 - 6FFFFF	Non-vol
700000 - 71FFFF	Timer
740000 - 77FFFF	Serial Interface
780000 - 7BFFFF	I/O ports on local bus
7C0000 - 7FFFFF	VXI interface
800000 - FFFFFF	User area

A list of internal routines available to the user are provided in Appendix A.

4.3.1 EPROM Organization

low memory #H0		- Vectors
	#H100	- Jump Table
	#H800	- Breadboard Code
high memory	#H10000	- User Code



The ROM contains vectors for interrupts, a table of routines that are callable from user code and the code for driving the interface, and support routines. ROM layout is as shown above, with a description of each section as follows.

The interrupt vectors are fixed in ROM, but provision has been made to allow the user to revector them.

Example:

> The command "setvect(26,#H10000);" sets the IRQ2 line to interrupt at address #H10000.

The jump table is at a fixed location in memory. The table contains jump instructions to a group of routines that are useful to the user. By incorporating a supplied "include" file, user code can call these routines as though they were in a library.

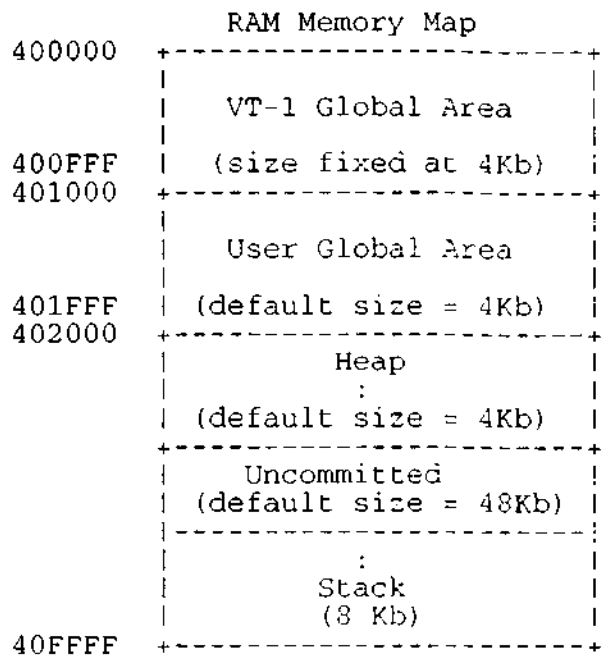
The breadboard code area contains support routines and the code for operating the interface.

The upper part of ROM space is available for user code. Racal Instruments can load user-supplied application code into this upper 64 Kb of the EPROM address space. The customer will supply Racal Instruments with the data files encoded in a suitable ASCII format, such as Motorola S-records.

4.3.2 RAM

RAM space is divided into the following general areas:

low memory	#H400000-400FFF	- Breadboard System RAM
	#H401000-programmable	- User Global Area
	Programmable	- I/O Queues
	Programmable	- Heap
high memory		- Stack



A total of 64Kb of RAM is available in the standard message-based interface. The lower 4Kb is dedicated system RAM, and is used by the interface for maintaining the operation of the VXIbus interface and user support routines.

The user global area starts at a fixed location. This space is intended for users who download code into the breadboard. The default size of the user "global" area is 4Kb, but its total size is programmable. This address space will always begin at 401000. The user may place global variables in this area and/or load application specific code for later execution.

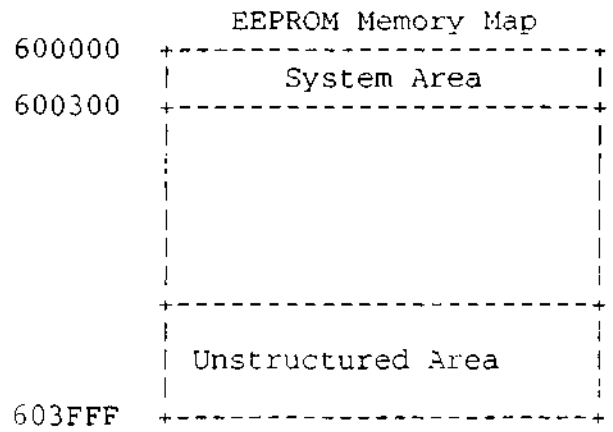
The I/O queues are established during power-up initialization. They are positioned above the user global area. The user can specify the size of the queues. The size is stated in the manual.

The heap is memory that can be allocated as requested. It floats above the I/O queue area. The heap grows up toward the stack area. The default size of the system heap is 4Kb by the total size programmable.

The stack area goes below the debugger area. It grows down toward the heap. The default size is also 4Kb, but the total heap size is programmable. The total message based software requires 8Kb of the stack area.

This means with a standard configuration, the user has over 48 Kb of RAM available.

4.3.3 EEPROM (Non-Vol)



Non-Vol memory can be modified under program control and retains the information even when power is removed. Non-Vol is used for many things in the breadboard. Some of the things that can be stored in Non-Vol are:

Required boards configuration information
User specified vectors
Initial I/O port values
Queue sizes
User data

The Non-Vol memory is divided into two areas. The first area is a fixed block of memory dedicated to system information. The second area is the user area. The user area can be divided into a structured area that uses "records", and an unstructured area where the user has complete flexibility.

Non-Vol records are stored as a linked list. Each record has common pieces of information as well as the actual information being stored.

The common information is:

Block Links
Record Type Identifier
Block Checksum
Description
Data Area

Block Links are links that allow for the maintenance of the Non-Vol's structure.

Record Type Identifiers contain flags that describe some of the capabilities for this block. One flag is used to determine whether the data from this block is allowed to be sent out

over the VXIbus on request. It provides primitive security capabilities. Another flag is used to identify this as a user or a breadboard block. There are two flags reserved for future use. Four bits are available for the user to identify different types of blocks.

The Block Checksum is used to verify integrity of the block. It is checked during power-up initialization.

Block Size is used for maintaining the Non-Vol structure. The user can store a description of each block. Commands can be used to cause this information to be sent over the VXI. The information to be stored goes in the data area. The type of information can be just about anything the user wants to store.

There is a set of commands available from the VXIbus to manipulate the Non-Vol memory. The standard Non-Vol memory size is 16K bytes. The system block size is about 3/4 K bytes.

4.3.3.1 The System Area

The system area maintains a variety of Non-Vol records for operation of the system.

The user can specify a board configuration by selecting a port, value and a mask. When that is specified, it is stored in Non-Vol. The system reads the board configuration at power-up, and will report if any configurations do not match.

Initial port values can be specified and stored in Non-Vol. These cause values other than the default jumper values to be setup shortly after power-up.

If the user wants to change some of the initial values of the VXI registers, they can be stored in Non-Vol. As the system powers up, these values will be placed in the appropriate registers (see Appendix B).

The user can specify a number of vectors for asynchronous events. These vectors can be stored in Non-Vol and setup as the system is powering up.

The user can specify the size of the input queue, the secondary input queue and the output queue. These sizes can be stored in Non-Vol and used to setup the queues at power-on initialization.

User code can be saved in Non-Vol so it can be used as the system is powered-up or as part of the main loop without down loading code each time.

The interface software uses the system block to hold information to customize the operation of the interface. Information saved in the system block is shown in the structure nv_sys in the nv.h header file. Here is a reproduction of that structure (For the actual structure for your version of the firmware, look at nv.h on the supplied diskette).

```
        struct nv_sys
{
unsigned long global_size;      /* Size of global area */
unsigned long pri_size;         /* Size of primary queue */
unsigned long sec_size;         /* Size of secondary queue */
unsigned long out_size:         /* Size of output queue */
unsigned long heap_size;        /* Size of heap */
unsigned long nv_save_size;     /* Size of Nonvol to save for code */
unsigned long vxi_reg_inuse;    /* LSB is for ID reg */
unsigned int vxi_id;                        /* VXI registers' initial values */
unsigned int vxi_dt;                        (See note.1)
unsigned int vxi_status;
unsigned int vxi_06;
unsigned int vxi_ptotocol;
unsigned int vxi_response:
unsigned int vxi_0c;
unsigned int vxi_data_low;
unsigned int vxi_10;
unsigned int vxi_12;
unsigned int vxi_14;
unsigned int vxi_16;
unsigned int vxi_18;
unsigned int vxi_1a;
unsigned int vxi_1c;
unsigned int vxi_1e;
unsigned int vxi_20;
unsigned int vxi_22;
unsigned int vxi_24;
unsigned int vxi_26;
unsigned int vxi_28;
unsigned int vxi_2a;
unsigned int vxi_2c;
unsigned int vxi_2e;
unsigned int vxi_30;
unsigned int vxi_32;
unsigned int vxi_34;
unsigned int vxi_36;
unsigned int vxi_38;
unsigned int vxi_3a;
unsigned int vxi_3c;
```

```
unsigned int vxi_3e;
unsigned int start_user;        /* Start in user parser? */
unsigned int term_vector;       /* EOI non-usage flag & terminator count */
char terminators[4];
void(* routine[64])();          /* Vectors */
                        /* 0 - user power on */
                        /* 1 - user parser */
                        /* 2 thru 47 correspond to 68000 */ (See note 2)
                        /* 48 - user software trigger */
                        /* 49 - reserved */
                        /* 50 - user parse error handler */
                        /* 51 - user clear handler */
                        /* 52 - user clock rollover handler */
                        /* 53 - user signal handler */
                        /* 54 to 63 reserved */
unsigned long nv_flags;         /* Misc. flags */
                        /* Bit 00 - ROM_VECTOR */ (see note 3)
                        /* Bit 01 - ROM_INIT */  (see note 4)
                        /* Bit 02 - Unused */
                        /* Bit 03 - Unused */
                        /* Bit 04 - Unused */
                        /* Bit 05 - Unused */
                        /* Bit 06 - Unused */
                        /* Bit 07 - Unused */
                        /* Bit 08 - Unused */
                        /* Bit 09 - Unused */
                        /* Bit 10 - Unused */
                        /* Bit 11 - Unused */
                        /* Bit 12 - Unused */
                        /* Bit 13 - Unused */
                        /* Bit 14 - Unused */
                        /* Bit 15 - Unused */
                        /* Bit 16 - Unused */
                        /* Bit 17 - 31 - Unused */
```

Notes:

1. The unsigned integers, vxi_id thru vxi_3e are described in the VXIbus Specifications (Appendix I of Revision 1.3).
2. The vectors 2 thru 47 are described in table 6.2 of the 68000 handbook.
3. Bit 00 - ROM_VECTOR, is used for going to initialization routines in ROM, at power-on.
4. Bit 01 - ROM_INIT, is used for setting up vectors and user parsers. Whatever is at location H10000 during power-on is used if this flag is set.

4.3.3.2 User's Area

The Non-Vol memory not used by the system block is available for the user's area. The user's area is divided into two sections. One section uses "records" to hold information. The other area is unstructured. The sixth item in the system block (unsigned long nv_save_size from previous nv list) determines how much Non-Vol is used for the "record" area, and how much is left unstructured. Routines in the firmware maintain "records" with links, identifiers, a checksum and a description. The beginning of a "record" must use the structure shown below (also from nv.h). Only the description and the data are not included in the structure. Records are useable only when programming in ROM.

```
struct nv_header
  {
  struct nv_header *link;
  long size;
  unsigned int checksum;
  char  block_id;
  char  descript_len;
  };
```

4.3.3.3 Initialization of Non-Vol

Because it is possible to make the unit not respond after power-up by storing values in Non-Vol, a method is provided to allow Non-Vol memory to be initialized.

NOTE

All information stored in Non-Vol will be lost!

To initialize Non-Vol, connect pin 1 to pin 2 on E1 during the power-up sequence. E1 is located at one corner of the interface board, near the P100 connector. Pin 1 is the square pad. Pin 2 is the pad directly above it (the closest pad in the E1 group to the center of U7/U13). After initializing, remove the connection. If the connection is not removed, non-vol memory will be initialized each time the unit is powered up.

4.4 PEEKING AND POKING THE 68000

The Message-Based Interface allows the user to "poke" and "peek" words and bytes to/from any location within the 68000's address space. The intent of these commands is to allow the user to exercise I/O ports and circuitry. The adventurous Message-Based Interface user will soon realize that programs can be "poked" uncommitted RAM or the user global area. The following C program illustrates this point. It will endlessly toggle a selected bit of a specified I/O port. It's intended to be run from the command line of the host system. It "pokes" a simple loop into the beginning of the user global area, and selects this simple routine as the "application parser" to be repeatedly called.

```
#include <stdio.h>
#include <ctype.h>

extern void exit(int);

/* each elem = "0x" + 4 hex digits + NULL */
char Instrs[][6+1] =
        {
        "0x207c", /* movea.l #<address>, a0 */
        "....", /* upper 16 address bits  */
        "....", /* lower 16 address bits  */
        "0x70ff", /* moveq #-1, d0 ; 64K loop iters */
/* loop */ "0x1210", /* move.b (a0), d1 */
        "0x0841", /* bchg #<bitnum>, d1 */
        "....", /* bit to toggle */
        "0x1081", /* move.b d1, (a0) */
        "0x51c8", /* dbf d0, loop */
        "0xfff6", /* offset back to 'loop' label */
        "0x4e75", /* rts */
        };

#define         NUM_INSTRUCTIONS (sizeof(Instrs)/(6+1))

void pokeNrunToggleBitCode(unsigned long addr,unsigned bitnum)
{
        char buf[100];
        int i, ud;
        const unsigned long targaddr = 0x00401000L;
        unsigned upper, lower;
        ud = ibfind ("VXITECH");
        upper = ((addr & 0xffff0000L) >> 16);
        lower =  (addr & 0x0000ffffL);
        sprintf(&Instrs[1], "0x%.4x", upper);
        sprintf(&Instrs[2], "0x%.4x", lower);
```

```
        sprintf(&Instrs[6], "0x%.4x", bitnum);

        for (i = 0; i < NUM_INSTRUCTIONS; i++)
                {
                sprintf (but, "poke (0x%.8lx, %s);\n",
                                                targaddr + i * 2, &Instrs[i]);
                ibwrt (ud, buf, strlen (buf));
                }
        sprintf (buf, "setparser (0x%.8lx);\n", targaddr);
        ibwrt (ud, buf, strlen (buf));
}

void usage(void)
{
        fprintf (stderr,"Usage: togbit -a <hex address> -b <bitnum>\n");
 }

void main(argc, argv) char *argv[];
{
        unsigned long addr = 0; /* Use these bogus values to do */
        unsigned bitnum = -1; /* some error checking later */

        /* Collect command-line args */
        while (argc > 1 && argv[1][0] == '-')
                {
                switch (toupper(argv[1][1]))
                        {
                        case 'A':
                                if (sscanf(argv[2], "%lx", &addr) != 1)
                                        {
                                        fprintf(stderr,
                                                "Illegal numerical format inaddress\n");
                                        usage(); exit(1);
                                        }
                                argc -= 2; argv += 2;   /* shift cmd line args */
                                break;

                        case 'B':
                                if (sscanf(argv[2], "%d", &bitnum) != 1)
                                        {
                                        fprintf(stderr,
```

```
                                                    "Illegal numerical format in
bitnum\n");
                                                usage(); exit(1);
                                                }
                                        argc -= 2; argv += 2; /* shift cmd line args */
                                        break;

                                default:
                                        fprintf(stderr, "Unknown option %s\n", argv[1]);
                                        usage(); exit(1);
                                        break;
                                }
                        }
                if (addr == 0 || bitnum < 0)
                        { usage(); exit(1); }

                pokeNrunToggleBitCode(addr, bitnum);
                exit(0);
        }
```

4.5 USERS APPLICATION SPECIFIC CODE

4.5.1 Loading Application Code

This section describes the methods of loading code into the Message-Based Interface.

4.5.2 S-Record Download

The Message-Based Interface allows the user to download a named block of executable code as a sequence of Motorola S-records. At the time the user links the application code, the target load address should be set either in the uncommitted RAM area, or in the Message-Based Interface user global area. As the Message-Based Interface receives each S-Record, it unpacks and loads the encoded data into the specified addresses. It is the user's responsibility to manage the contents of the uncommitted RAM area and the user global area. The Message-Based Interface will dutifully overwrite any existing section of the user global area while unpacking S-records. The number of individual blocks that may be downloaded is only constrained by the physical size of the uncommitted RAM area and the user global area.

A C-style identifier is associated with each block of downloaded code. The Message-Based Interface stores this identifier into its global symbol table for later reference.

The precise syntactical mechanics of this operation are spelled out in the section on Message-Based Interface command syntax. The following example will illustrate the process:

```
/*
        This function downloads a single RTS instruction into the first word of the
        Message Based Interface user global area.  The identifier "exampleRTS"
        is given to this executable block. Subsequent to S-record download, a
        Message-Based Interface command is sent to invoke the named function
        we've just downloaded, e.g. "exampleRTS".

*/
void downloadAndInvokeRTS(int ud /* unit descriptor */)
        {
        extern int strlen(const char *s);
        static char *downloadCmd = "download exampleRTS();";
        static char *Srecords[] =
                {
                "S00600004844521B", /* header record */
                "S2054010004E75E7", /* an RTS instr  @ 0x401000 */
                "S804401000AB"  /* transfer addr = 0x401000 */
                };
        #define NRCDS (sizeof(Srecords)/sizeof(char*))
        static char *invokeCmd = "exampleRTS();"
        int ud, i;

        /* Open the Message Based Interface */
        ud = ibfind ("VXITECH");

        /* Tell the Interface you're about to download a named block */
        ibwrt (ud, downloadCmd, strlen(downloadCmd));

        /* Download a single RTS instruction */
        for (i = 0; i < NRCDS; i++)
                ibwrt (ud, Srecords[i], strlen(Srecords[i]));

        /* Tell the Interface to invoke the named block */
        ibwrt (ud, invokeCmd, strlen(invokeCmd));
        }
```

Below is a program that can be found on the accompanying disk that downloads an S-Record file onto the 7064. An example of a S-Record is also included in file VTI_HEX.

```
COMMON SHARED /NISTATBLK/ IBSTAT%, IBERR%, IBCNT%
DECLARE SUB IBRD (BD%, RD$)
DECLARE SUB IBWRT (BD%, WRT$)
DECLARE SUB IBFIND (BDNAME$, BD%)

CLS
PRINT "Sample program to download an S record file into the VTI."
PRINT "Copyright 1990 by VXI Technology, Inc. All Rights Reserved."
PRINT

CALL IBFIND("VXITECH", BD%)
  IF BD% < 0 THEN
    PRINT "Device 'VXITECH' not found."
    STOP
  END IF

INPUT "What file would you like downloaded (include extension)";F$
OPEN F$ FOR INPUT AS #1

INPUT "What function name would you like assigned to it"; AFN$

WRT$ = "download " + AFN$ + "(); "
CALL IBWRT(BD%, WRT$)

BLIP = 0
DO UNTIL EOF(1)
   INPUT #1, WRT$
   CALL IBWRT(BD%, WRT$)
   BLIP = BLIP + 1
   IF BLIP >= 50 THEN
      BLIP = 0
      PRINT "*";
   ENDIF
LOOP

CLOSE #1
END
```

4.5.3 Executing Application Code

Once a named block of code has been loaded into the Message-Based Interface using the 'DOWNLOAD' command, it can be invoked like any other built-in Message-Based Interface command. The style is reminiscent of C-language function calls. Referring to the example program shown in Section 4.1.1, when the controller sends the string "exampleRTS();", the Message-Based Interface will execute the user-supplied function called "exampleRTS".

4.6 APPLICATION SPECIFIC COMMAND PARSER

It is possible for the user to supply a secondary, application-specific command parser. The 7064 has a single-threaded control flow architecture, and it passes control to the user's command parser under two conditions:

1. While Waiting For Input. Rather than doing a 'busy wait' on an empty input queue, the 7064 command parser will make a function call to the user parser and allow it to run.

   NOTE

   Under these conditions, there may be no input for the user's parser. This tactic affords the user's parser some extra time for computations and other functions.

2. When Explicitly Directed By The "USR_SCAN();" Command. The 7064 command parser monitors the character input stream, and parses and interprets its own command set. When the "USR_SCAN();" command is recognized in the input stream, the 7064 command parser copies subsequent input into a secondary queue and calls the user's parser, which is then free to fetch the characters from the secondary queue and interpret them. In this way, the user may freely mix 7064 and application specific command sets in the same input stream.

While running in this "user command" mode, the 7064 command parser continues to monitor the input stream, copying characters into the secondary queue, and invoking the user's parser until it recognizes the string "VT1_SCAN();". This string is a directive to switch back to the normal 7064 command processing mode.

By delimiting 7064 commands and application specific commands by the strings "USR_SCAN();" and "VT1_SCAN();", the user is able to mix command sets in the same input stream, and switch parsing modes at selected points.

Note that both "USR_SCAN();" and "VT1_SCAN();" should appear alone on a line, terminated by a new line or an EOI.

4.6.1 Selecting a Block of Code as the Application Command Parser

As mentioned previously, it is possible for the user to load an application-specific command parser. This is accomplished using the "DOWNLOAD" command described above. The "SETPARSER" command informs the Message-Based Interface a block of code is to be run as the application parser. The "SETPARSER" operation is available both as a Message-Based Interface command and as a callable system service. The syntax of the SETPARSER command is:

SETPARSER (<address>)

where <address> is the location into which the parser has been loaded.

The function prototype for the system service is:

```
extern void setparser(const unsigned long address);
```

4.6.2 Control Flow Models

As previously described, the application parser is called in two ways:

1. While waiting for input, rather than "busy wait" on an empty input queue while the machine is idle, the Message Based Interface will repeatedly call the application parser.

2. As explicitly directed. The user may explicitly command the Message-Based Interface to redirect the subsequent input stream to the application-specific parser (The exact details of this operation are described in the section on Message-Based Interface command syntax).

4.6.3 Character I/O Available to the Application Command Parser

The "Raw" input is available to the application parser in a secondary input queue.

more data is available in the secondary input queue.

NOTE

To avoid system deadlock, the application parser should never do a "busy wait" on an empty queue. It should return to the calling function. It will be called again as soon as more data is available in the secondary input queue.

Here is a summary of the queue access functions:

```
#include "vt1_rou.h"

/*      Get next "raw" char from secondary input queue.h.
        Returns  NO_CHAR if queue is empty.                 */

        extern int vt1_getchar_raw(void);

/*      Get next "stripped" char from secondary input queue.
        The upper 8 bits have been masked off.
        Returns NO_CHAR if queue is empty.                  */

        extern int vt1_getchar(void);

/*      Returns a character "ch" to the secondary input queue; the state of the upper 8
        status bits is maintained.
        Only a single character may be pushed back into the queue.
*/
        extern int vt1_ungetc(int ch);

/*      Returns a count of the number of characters available in the secondary input
        queue.                                              */

        extern long in_stat(void)

/*      Returns non-zero (TRUE) if an EOI was asserted.   */

        extern int eof(void)
```

4.6.4 Sample Application Parser

```
#include "vt1_rou.h"
void myParser(void)
{
        int c;
        /*
                Echo characters received in the secondary queue to the output queue;
                maintain EOI status on last char.
        */
                while ((c = getchar()) != NO_CHAR)
                vt1_putchar(c, eof() ? 1 : 0);
 }
```

4.7 ACCESSING MESSAGE-BASED INTERFACE SYSTEM SERVICES

The Message-Based Interface provides the following general classes of system services. The functions that implement these services are part of the Message-Based Interface firmware, and are accessible to user-created application code through a jump table. A C include file provides the function prototypes necessary to compile C programs that make use of these services.

Low-Level Port I/O
setbit(), rstbit(), inbit(), outbit()
inport(), inportb(), outport(), outportb()
pulsehi(), pulselo()

Low-Level Port I/O with card-relative addressing
csetbit(), crstbit(), cinbi(), coutbit()
cinport(), cinportb(), coutport(), coutportb()
cpulsehi(), cpulselo()

Triggering
triggen(), triggin(), trigout()

Timer Management
delay(), gettime(), settime(), sleep()

Non-Vol Management
delete_nv(), locate_nv(), store_nv(), retrieve_nv()

Primary Input Queue Management
eof_pri(), getchar_pri(), in_stat_pri(), getchar_raw_pri(), ungetchar_pri()

Secondary Input Queue Management
vtl_ungetc(), putchar_sec(), vtl_getchar(), in_stat(), eof()

Output Queue Management
out_stat(), vtl_putchar(), vtl_puts()

Interrupt Management
setvect(), disable(), enable(), restoreintlevel(), setintlevel(), geninterrupt(), getvec()

Error Handling
signal_err()

Memory Management
vt1_calloc(), vt1_cfree(), vt1_free(), vt1_malloc(), peek(), peekb(), poke(), pokeb()

Output
base(), sscanf(), sprintf(), vt1_out()

4.7.1 Compiling and Linking for Message-Based Interface System Services

The following C file should be included by any user application that makes use of the Message-Based Interface system services. This is the complete list of software commands available to the user at time of publication. These are described in detail in Appendix A.

vt1_rou.h /* Function prototypes */

```
/*
Definitions of routines and variables available in the VT-1 ROM system.
*/
#ifndef VT1_ROU
#define VT1_ROU

#define NULL 0L
#define NO_CHAR 0x200
typedef unsigned int uint;
typedef unsigned long ulong;
typedef void (* Interrupt)(void);
typedef void (* Inter)(void);

struct  time
  {
  unsigned char  ti_hour;    /* Hours */
  unsigned char  ti_min;     /* Minutes */
  unsigned char  ti_sec;     /* Seconds */
  unsigned int   ti_milli;   /* Thousandths of seconds */
  unsigned int   ti_micro;   /* Millionths of seconds */
  };

struct  raw_time
  {
  ulong seconds;
  ulong micros;
  };

extern int     abs (int i);
extern int     atoi (char *string);
extern long    atol (char *string);
```

```
extern void    base (uint base_value);
extern uint    cinbit (uint card, uint port, uint bitnum);
extern uint    cinport (uint card, uint cport);
extern uint    cinportb (uint card, uint cport);
extern void    clear_done (void);
extern void    coutbit (uint card, uint cport, uint bit_number, uint value);
extern void    coutport (uint card, uint cport, uint value);
extern void    coutportb (uint card, uint cport, uint value);
extern void    cpulsehi (uint card, uint cport, uint bit_number);
extern void    cpulselo (uint card, uint cport, uint bit_number);
extern void    crstbit (uint card, uint cport, uint bit_number);
extern void    csetbit (uint card, uint cport, uint bit_number);
extern void    delay (ulong microseconds);
extern int     delete_nv (void *nv_blkptr);
extern void    disable (void);
extern void    enable (void);
extern int     cost (void);
extern int     cost_pri (void);
extern void    error_handler (uint error_description, void *argp);
extern char  * error_msg_lookup (uint error_description);
extern void    event (uint event_value);
extern void    execerror (uint error_description, ...);
extern char  * get_curr_input_line (void);
extern int     getchar_pri (void);
extern int     getchar_raw (void);
extern int     getchar_raw_pri(void);
extern void    gettime (struct time *timep);
extern ulong   gettimel (void);
extern Inter   getvect(int interruptno);
extern ulong   in_stat (void);          /* Input queue number of chars */
extern ulong   in_stat_pri (void);      /* Primary input queue chars */
extern uint    inbit (uint port, uint bit_number);
extern char  * index (char *src, char c);
extern uint    inport (uint port);
extern uint    inportb (uint port);
extern int     isalnum (int c);
extern int     isalpha (int c);
extern int     iscntrl (int c);
extern int     isdigit (int c);
extern int     isgraph (int c);
extern int     islower (int c);
extern int     isprint (int c);
extern int     ispunct (int c);
extern int     isspace (int c);
extern int     isupper (int c);
```

```
extern int     isxdigit (int c);
extern long    labs (long i);
extern void  * locate_nv (int rec_type, int sequence);
extern char  * memccpy (char *dest, char *src, char c, int num);
extern char  * memchr (char *src, char c, int num);
extern int     memcmp (char *src1, char *src2, int num);
extern char  * memcpy (char *dest, char *src, int num);
extern char  * memset (char *dest, char c, int num);
extern void    nvcheckport (uint port, uint mask, uint value);
extern void    nvnocheck (void);
extern void    nvnoflags (void);
extern void    nvnoset (void);
extern void    nvnovxireg (void);
extern void    nvsetflag (uint flag);
extern void    nvsetport (uint port, uint value);
extern void    nvsetvect (int vector_num, void(*isr)());
extern void    nvsetvxireg (uint register_num, uint value);
extern void    nvsizeglobal (ulong size);
extern void    nvsizeheap (ulong size);
extern void    nvsizeinq (ulong size);
extern void    nvsizenvcode (ulong size);
extern void    nvsizeoutq (ulong size);
extern void    nvsizesecq (ulong size);
extern void    nvstartinuser (uint yes);
extern void    nvstorecode (void *source, void *dest, ulong length);
extern ulong   out_stat (void);          /* Number of chars in output queue */
extern void    outbit (uint port, uint bit_number, uint value);
extern void    outport (uint port, uint value);
extern void    outportb (uint port, uint value);
extern uint    peek (int * address);
extern char    peekb (char * address);
extern void    poke (int * address, int value);
extern void    pokeb (char * address, char value);
extern void    pulsehi (uint port, uint bit_number);
extern void    pulselo (uint port, uint bit_number);
extern int     putchar_sec(int c);
extern int     rand ();
extern struct  raw_time * point_time (void);
extern void    rcopy (void *table);
extern void    restoreintlevel (uint old_level);
extern void    retrieve_nv (void *dest, void *nv_blkptr);
extern char  * rindex (char *src, char c);
extern void    rstbit (uint port, uint bit_number);
extern void    setbit (uint port, uint bit_number);
extern void    set_error_handler (void (*errfunc)(uint error_desc, void *argp));
```

```
extern uint    setintlevel (uint new_level);
extern void    setparser (ulong address);
extern void    settime (struct time *timep);
extern void    setvect (int interruptno, void (*isr) ());
extern void    signal_err(void);
extern void    sleep (uint seconds);
extern int     sprintf (char *buffer, char *format, ...);
extern void    srand (int seed);
extern int     sscanf (char *buffer, char *format, ...);
extern int     store_nv (int rec_type, char *dscrpt, void *dataptr, long size);
extern char  * strcat (char *dest, char *src);
extern char  * strchr (char *src, char c);
extern int     strcmp (char *src1, char *src2);
extern char  * strcpy (char *dest, char *src);
extern int     strcspn (char *src, char *list);
extern int     stringcmpi (char *src1, char *src2);
extern int     strlen (char *src);
extern char  * strncat (char * dest, char * source, uint max);
extern int     strncmp (char *src1, char *src2, int num);
extern char  * strncpy (char *dest, char *src, int num);
extern char  * strpbrk (char *src, char *list);
extern char  * strrchr (char *src, char c);
extern int     strspn (char *src, char *cmp);
extern char  * strtok (char *src, char *sep);
extern void    swab (char *src, char *dest, int nbytes);
extern int     tolower (int c);
extern int     toupper (int c);
extern void    triggen( void);
extern void    triggin (uint line);  /* 0:7 is line, 8 is disable */
extern void    trigout (uint line);  /* 0:7 is line, 8 is disable */
extern int     ungetchar_pri(int c);
extern void  * vtl_calloc (long count, long size);
extern void    vtl_cfree( void *block);
extern void    vtl_free (void *block);
extern int     vtl_getchar (void);
extern void  * vtl_malloc (long size);
extern void    vtl_out(uint value, uint end);
extern void    vtl_outl(ulong value, uint end);
extern int     vtl_putchar (uint ch, uint end);
extern int     vtl_puts (char *s, uint end);
extern int     vtl_ungetc (int ch);

#endif   /* VTI_ROU */
```

This section is a partial summary of the system services provided by the Message-Based Interface. At the time of this writing, the number of services provided is growing. The complete list is to be found in the C include file "vtl_rou.h". This section also describes how users can compile and link to Message-Based Interface system services.

4.7.2 Linking - How The Jump Table Works

The file "jumptabl.68k" is a 68000 assembly source file defining the organization and absolute address of the Message-Based Interface jump table. When this file is assembled and linked into the user's application, it binds all references to Message-Based Interface system services. When a user's application calls a system service, like "rstbit()", the call is bound to an entry in the jump table. Each entry in the jump table contains a single JMP instruction that jumps to the selected service function.

4.8 PUTTING IT ALL TOGETHER

Refer to Figure 1. This is a block-diagram of a Message-Based Interface system into which an application command parser and a suite of application functions has been loaded. The boxes represent major functions. The arrows show control flow, e.g "A-->B" is read as function A calls function B. The Message-Based Interface system services are all enclosed inside a dotted border. The jump table is shown and illustrates how user applications are linked to Message-Based Interface system services. Each line of control flow is labeled. The individual flows are discussed below.

Flow 1: The Message-Based Interface command parser directly calls the system services. ASCII command strings like "RSTBIT" are mapped directly into function calls.

Flow 2: The Message-Based Interface command can call a named block of code that's been downloaded.

Flow 3: The Message-Based Interface command parser will call an application specific parser when (a) it is idling, waiting for new input, or (b), when it receives an explicit command to do so.

Flow 4: The application specific parser can also make direct calls to other blocks of application code.

Flow 5: Application code invokes a Message-Based Interface system service. It really calls an entry in the system jump table.

Flow 6: A JMP to the absolute address of the selected service.

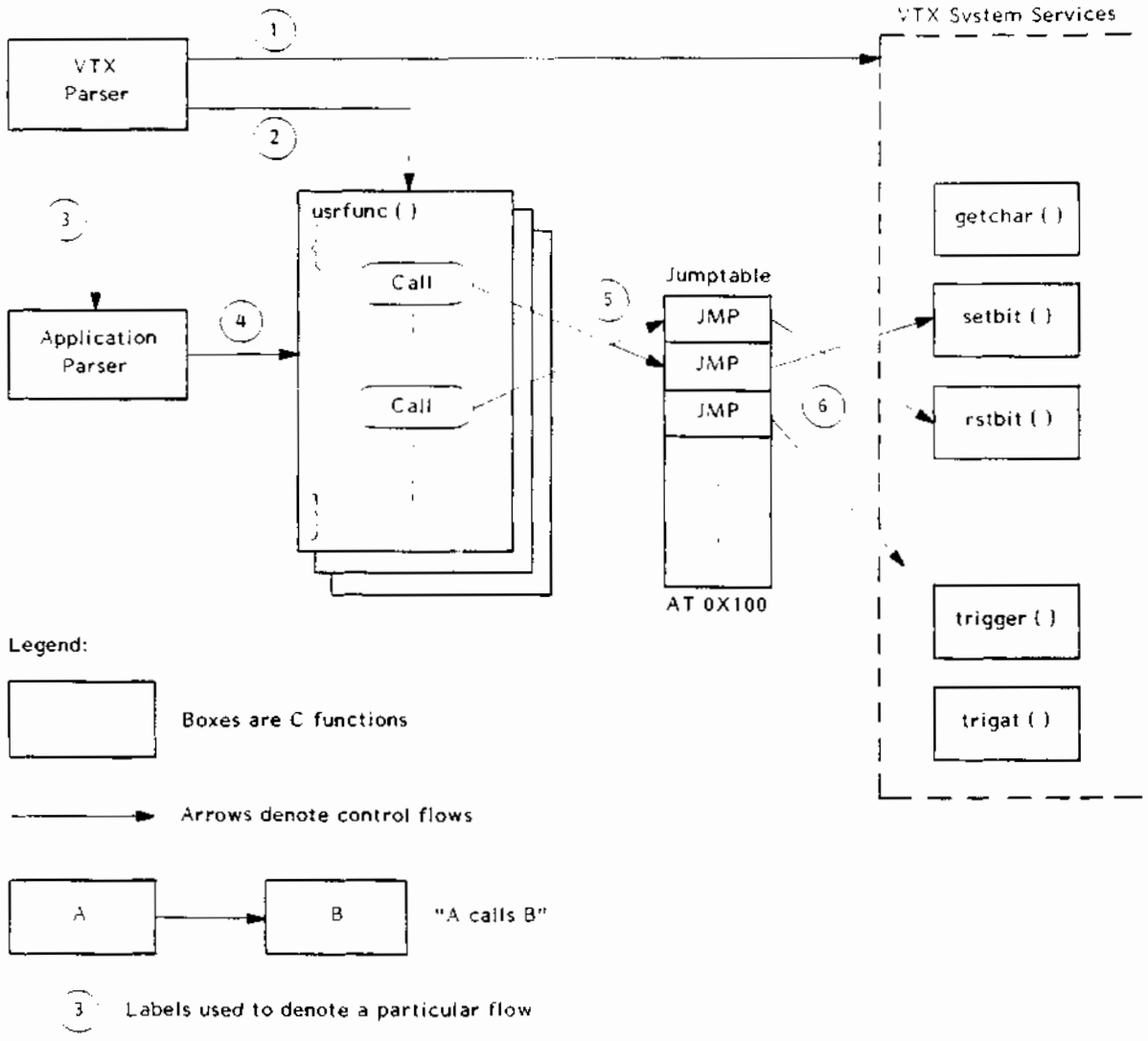

VTX Parser
VTX System Services
1
2
3
usrfunc ( )
Call
Call
Application Parser
4
5
Jumptable
JMP
JMP
JMP
6
AT 0X100
getchar ( )
setbit ( )
rstbit ( )
trigger ( )
trigat ( )
Legend:
Boxes are C functions
Arrows denote control flows
A
B
"A calls B"
3
Labels used to denote a particular flow

Figure 4.1

SECTION 5 REGISTER-BASED BREADBOARD APPLICATIONS

5.1 INTRODUCTIONS

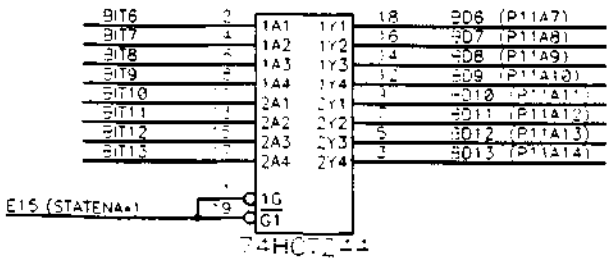
5.1.1 The 7064-20 has many features and great flexibility. This information describes some of the features, and give examples of how typical circuits can be implemented.

5.2 REGISTERS

5.2.1 All VXIbus devices have a set of registers that appear in the A16 address space. Each device is given 64 bytes of A16 address space for its registers. Register-based devices have four 16-bit registers that are normally implemented. Three of these registers are read registers. They are ID Register, Device Type Register, and Status Register. There is one Write Register - the Control Register. There is an optional Read/Write Register called the Offset Register. All of the rest of the A16 address space beyond the lower four registers is devoted to Device dependent registers. These registers can be used for any desired purpose.

5.2.1.1 Status Register

5.2.1.1.1 There are 12 bits in the Status register that are called device dependent. This means they can be used by the instrument designer for whatever purpose is desired. The register-based breadboard implements four of these device dependent bits. These bits are controlled by the levels at E18, E19, E20 and E21. The designer can also control the PASSED and READY bits in the Status register by controlling the levels of E16 and E17. The MODID bit is automatically controlled by the MODID bit on the backplane. The A24/A32 bit is automatically controlled by the A24/A32 enable bit in the control register. If the device designer wants to implement the additional device dependent bits (6 through 13), that is easily accomplished with one additional IC. Here is an example circuit that shows how it can be done:



VXI Status Register Expansion

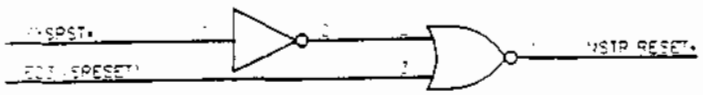
5.2.1.2 Control Register

5.2.1.2.1 The Control register is reset at power-up by a buffered version of the SYSRESET signal from the backplane (SYSRST*). The breadboard provides the reset bit out of the Control register at E23. This can be used to drive other circuitry that is dependent on this bit.

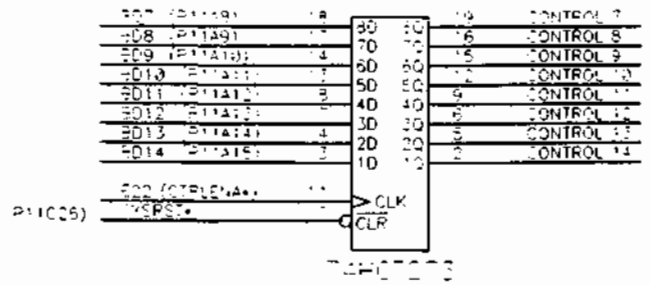
NOTE

This is not the same as the SYSRESET bit from the backplane. It is a bit the system controller uses to force a device into a reset state (the SYSRST* signal is provided at P11C26).

5.2.1.2.2 Typically, an instrument would be reset by either SYSRST* being low or the Control register reset bit being high. See the example circuit below. Another bit in the Control register is SYSFAIL INHIBIT. This bit is found at E24. The SYSFAIL INHIBIT is used by the system controller to disable driving the SYSFAIL line on the backplane even though the local board might have a fail condition. The most significant bit defined in the Control register is A24/A32 ENA. This bit is available at E30. This is generally used to tell a device whether to allow accesses in memory spaces other than A16. This bit would be used if A24 or A32 VXI memory was going to be put on the instrument. The breadboard also has five device dependent bits available (bits 2 through 6) from the Control register which appear at E25 through E29. The instrument designer can use these bits for whatever purpose is desired. Writes to the Control register affect these bits. If even more bits are desired, the designer can use a single chip to implement the rest of the device dependent bits of the Control register:



INSTRUMENT RESET



VXI CONTROL REGISTER EXPANSION

5.2.1.3 ID Register

5.2.1.3.1 The breadboard implements all bits of the ID register. The bits are controlled by S2 and S3. The switches and the bits they control are shown in the section of the manual on switch settings. These bits are used to identify the manufacturer, the address space and the device class. Manufacturer ID numbers are obtained from the VXIbus consortium. Address space indicates the addressing modes of the device's operational registers (e.g., A16 only is indicated by both bits set high). Device class refers to the type of instrument this is (e.g., register-based is indicated by both bits set high). Refer to the section in the manual on switch settings and the VXIbus standard for detailed information.

5.2.1.4 Device Type Register

5.2.1.4.1 The breadboard implements all bits of the Device Type register. The bits are controller by S4 and S5. The switches and the bits they control are shown in the section of the manual on switch settings. These bits are used to identify the instrument model and tell how much memory space is required by this instrument. The manufacturer is at liberty to set the model to almost any code desired.

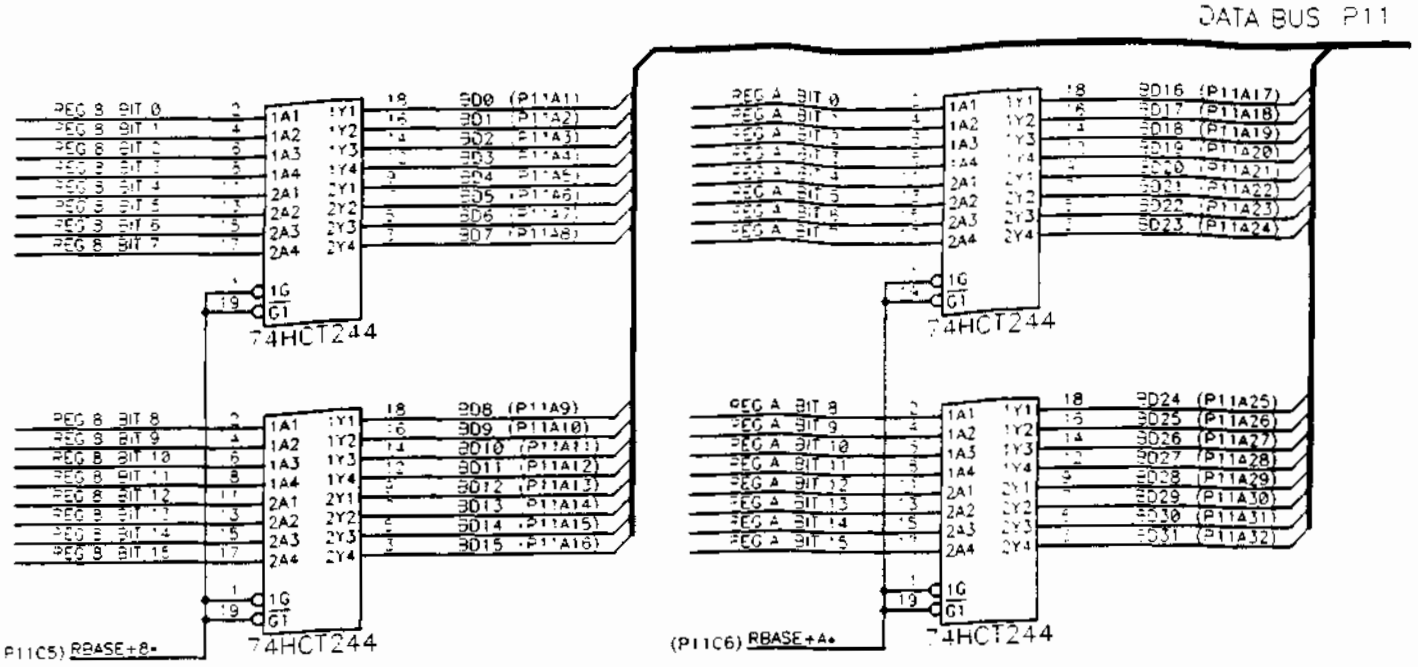
NOTE

Model codes 0 to 255 are reserved for slot 0 devices, and in most cases, should not be used.

5.2.1.4.2 Required memory is indicated by the upper four bits of the device type register. If the instrument is an A16 only device, the upper four bits are used as an extension of the model number. Refer to the schematic and the VXIbus standard for detailed information .

5.2.1.5 Additional Read Registers - Internal Decoding

5.2.1.5.1 Decoding is provided for some additional read registers. Because each decoded control signal is for 16 bits and the internal bus is 32 bits wide, the user must pay attention to which group of 16 data bits to connect to any added registers. Controls RBASE+8 and RBASE+C should be used with BD0 through BD15. Controls RBASE+6, RBASE+A and RBASE+E should be used with BD16 through BD31. Controller accesses of 16 and 32 bits are automatically handled. The decoded control signals appear at P11, the user interface connector. Here is an example of how some additional registers can be added:

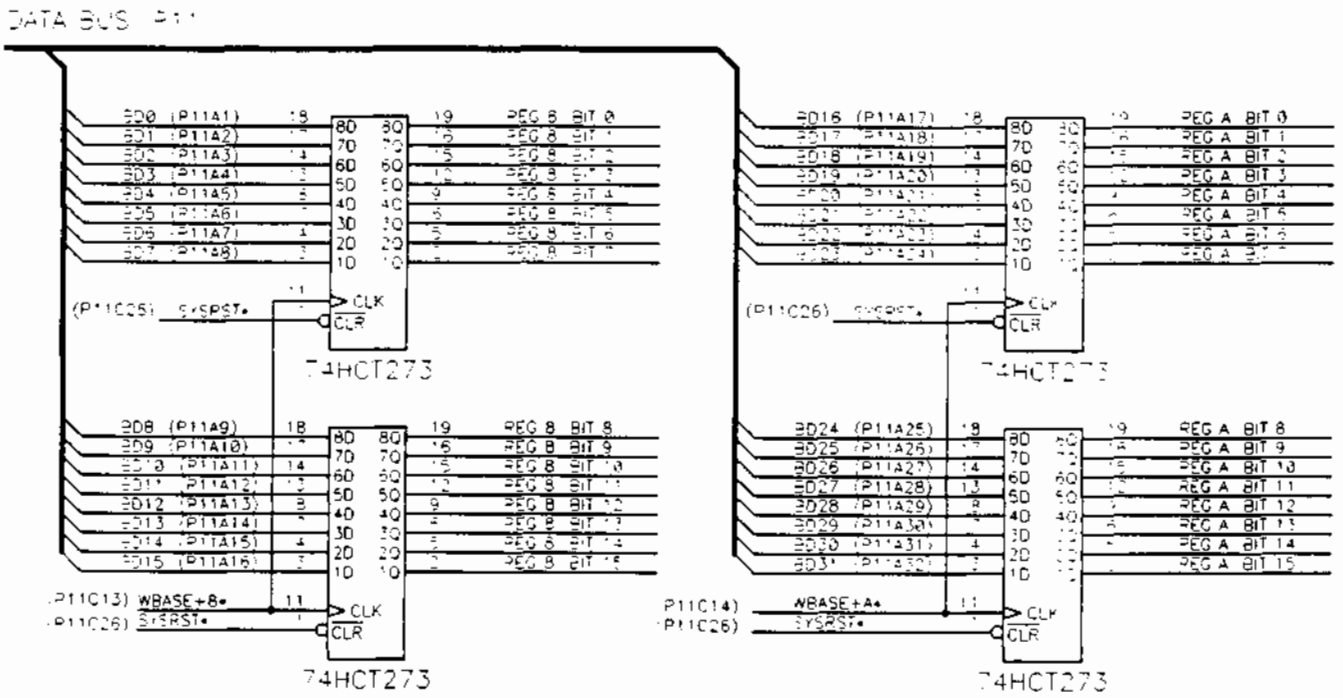


Additional Read Registers - Internal Decoding

5.2.1.6 Additional Write Registers

5.2.1.6.1 Decoding is provided for some additional write registers. Because each decoded control signal is for 16 bits and the internal bus is 32 bits wide, the user must pay attention to which group of 16 data bits to connect to any added registers. Controls WBASE+0, WBASE+8 and WBASE+C should be used with BD0 through BD15. Controls WBASE+2, WBASE+6, WBASE+A and WBASE+E should be used with BD16 through BD31. Controller accesses of the 32 bits are automatically handled. The decoder control signals appear at P11, the user

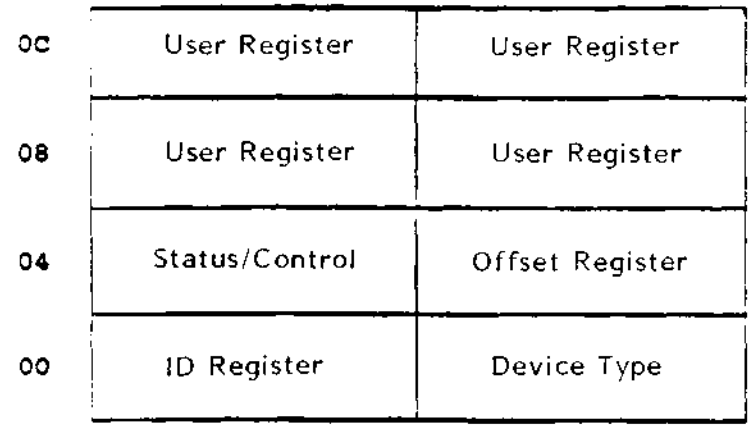
interface connector. Here is an example of how some additional write registers can be added:



Additional Write Registers - Internal Decoding

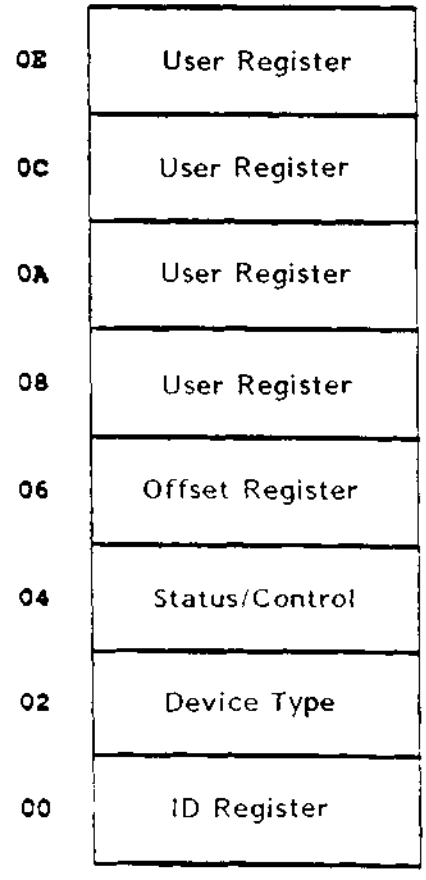
7064-20 Register Map

32 Bit Accesses



32 Bit Words

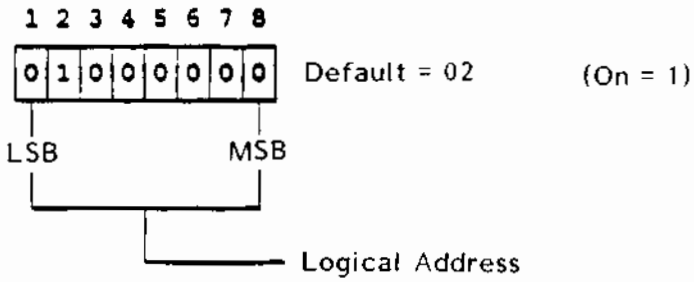
16 Bit Accesses



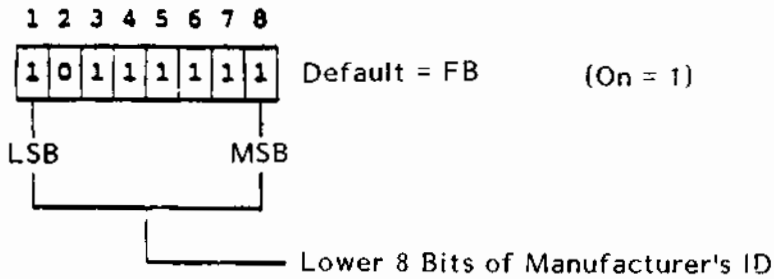
16 Bit Words

7064-20 Switches

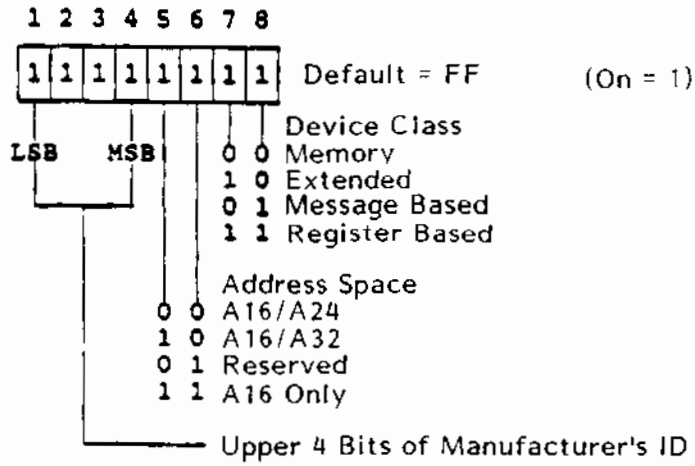
S1 Logical Address



S2 ID Register (Lower Half)

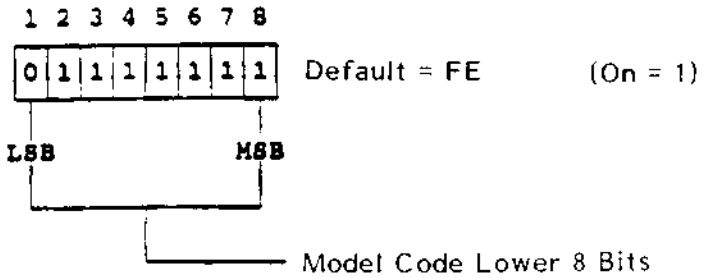


S3 ID Register (Upper Half)

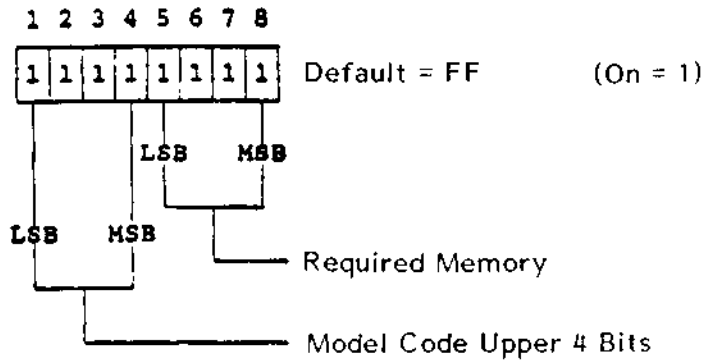


7064-20 Switches (continued)

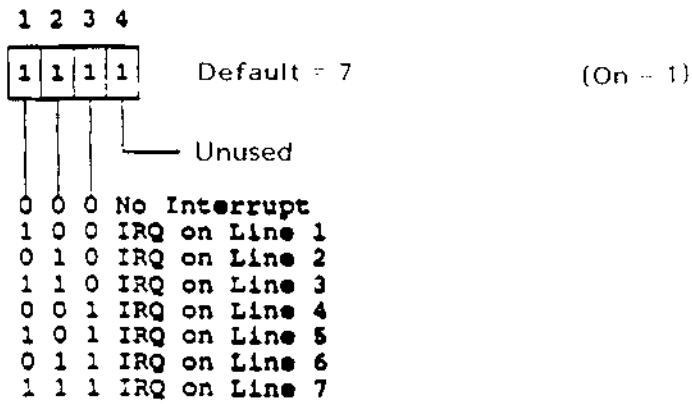
S4 Device Type Register (Lower Half)



S5 Device Type Register (Upper Half)



S6 Interrupt Request Line



SECTION 6 MAINTENANCE

6.1 TEST PROCEDURE

The test procedure described is used as the 7064 system verification procedure, and incorporates visual and functional tests (software driven).

6.2 REQUIRED EQUIPMENT

1. IBM PC or compatible with National Instruments GPIB-488 interface card.
2. VXIbus Mainframe (Racal Instruments Model 1261) and GPIB-VXIbus slot 0 Resource Manager (Racal Instruments Model 1260-00B).
3. 7064 with Message-Based Interface installed.

6.3 TESTS

6.3.1 Visual Tests

1. Verify the Message-Based Interface is installed and screwed down to the breadboard module.
2. Confirm the 7064 module address (not logical address) is set to "00", and jumper JP1 is positioned between pins 2 and 3.
3. Slide the 7064 module in and out of the VXIbus mainframe, and verify proper fit and installation.

6.3.2 Functional Tests

The listing of the test program written in BASIC has been provided. This program performs the following functional tests:

1. Board address test
2. Reset test
3. Output bit tests

The test program is menu driven and prompts the user for input.

6.3.2.1 Board Address Test

The board address test is performed by setting SW1 and doing a read/write at the proper address, and verifying the correct response: then doing a read/write at an incorrect address and verifying the correct response.

This test is repeated for each bit of SW1. In the performance of this test, address lines LA0 through LA14 are tested.

6.3.2.2 Reset Test

This test checks the RSTCLK line and associated hardware. A write/read is performed, and the 7064 is reset. A read is executed to verify reset has occurred.

6.3.2.3 Output Bit Test

This test performs write/read operations of the following patterns: A5 hex, 5A hex, walking high, bit and walking low bit pattern.

6.4 7064 TEST SOFTWARE

REM Copyright VXI Technology, Inc. 1990, All Rights Reserved

```
REM FILE: VT1TEST.BAS
' $INCLUDE: 'c:\gpib-pc\QBDECL4.BAS'
CLS
t = 1
noerror = 0
t$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
COLOR 14, 1, 0
KEY OFF
KEY 2, "ADRES"
KEY 3, "RESET"
KEY 4, "BIT"
KEY 5, "EXIT"
KEY ON
MAINMENU:
CLS
t = 1
LOCATE 2, 26: PRINT "7064 BREADBOARD TEST SOFTWARE"
COLOR 31, 0: LOCATE 22, 1: PRINT "SELECT DESIRED TEST BY USING THE
SOFTKEYS AS LABELED"
COLOR 14, 1, 0
LOCATE 8, 2: PRINT " 1.  CARD ADDRESS TEST..... SELECT F2"
LOCATE 9, 2: PRINT " 2.  MODULE RESET TEST..... SELECT F3"
LOCATE 10, 2: PRINT " 3.  OUTPUT BIT TEST..... SELECT F4"
LOCATE 11, 2: PRINT " 4.  EXIT TO DOS..... SELECT F5"
ON KEY(2) GOSUB ADDRESS: KEY(2) ON
ON KEY(3) GOSUB RESETTEST: KEY(3) ON
ON KEY(4) GOSUB OUTPUTBIT: KEY(4) ON
ON KEY(5) GOSUB STOPPER: KEY(5) ON
DO
LOOP WHILE t = 1
IF t = 0 THEN GOTO MAINMENU
'
ADDRESS:
KEY OFF
CLS
```

```
noerror = 0
LOCATE 2, 26: PRINT "CARD ADDRESS TEST"
FOR bits = 0 TO 4
IF bits = 0 THEN bit$ = "all bits OFF": digits = 0
IF bits = 1 THEN bit$ = "bit 1 ON, all others OFF": digits = 8
IF bits = 2 THEN bit$ = "bit 2 ON, all others OFF": digits = 4
IF bits = 3 THEN bit$ = "bit 3 ON, all others OFF": digits = 2
IF bits = 4 THEN bit$ = "bit 4 ON, all others OFF": digits = 1
LOCATE 12, 10: PRINT "Verify Card Address switch SW1 is set to "; bit$
GOSUB pauseroutine
CLS
LOCATE 2, 26: PRINT "CARD ADDRESS TEST"
GOSUB findit
FOR card = 0 TO 15
wrt$ = "coutportb(" + STR$(card) + ",0,#H22);"
wrt1$ = "cinportb(" + STR$(card) + ",0);"
CALL ibwrt(bd%, wrt$)
CALL ibwrt(bd%, wrt1$)
CALL ibrd(bd%, t$)
CLS
LOCATE 2, 26: PRINT "CARD ADDRESS TEST"
IF LEFT$(t$, ibcnt%) <> "255" THEN
IF STR$(digits) <> STR$(card) THEN GOSUB crderror
END IF
NEXT card
NEXT bits
CLS
LOCATE 2, 26: PRINT "CARD ADDRESS TEST"
LOCATE 10, 10: PRINT "Set all bits to OFF. "
IF noerror = 0 THEN LOCATE 20, 10: PRINT "Card Address Test Passed"
IF noerror = 1 THEN LOCATE 20, 10: PRINT "Card Address Test Failed"
GOSUB pauseroutine
t = 0
RETURN

RESETTEST:
KEY OFF
CLS
LOCATE 2, 26: PRINT "MODULE RESET TEST"
LOCATE 12, 10: PRINT "With power off, verify JP1 is between pins 1 and 2"
LOCATE 13, 10: PRINT "Turn power on"
GOSUB pauseroutine
CLS
```

```
LOCATE 2, 26: PRINT "MODULE RESET TEST"
GOSUB findit
GOSUB basevalue
E = 0
FOR port = 0 TO 11
wrt$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
' PRINT "PORT"; port, t$
IF LEFT$(t$, ibcnt%) <> "#B11111111" THEN GOSUB rderror
NEXT port
'LOCATE 20, 10: PRINT "Verify all PORTS and BITS HIGH"
'GOSUB pauseroutine
' CLS
E = 1
FOR port = 0 TO 11
wrt$ = "outportb(" + STR$(port) + ",0);"
wrt1$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibwrt(bd%, wrt1$)
CALL ibrd(bd%, t$)
'PRINT "PORT"; port, t$
IF LEFT$(t$, ibcnt%) <> "#B0" THEN GOSUB rderror
NEXT port
' LOCATE 20, 10: PRINT "Verify all PORTS and BITS LOW"
' GOSUB pauseroutine
' CLS
' LOCATE 13, 10: PRINT "Turn power OFF and then ON"
' GOSUB pauseroutine
' CLS
GOSUB findit1
wrt$ = "SysReset"
CALL ibwrt(bd%, wrt$)
' GOSUB basevalue
SLEEP 7
GOSUB findit
GOSUB basevalue
E = 0
FOR port = 0 TO 11
wrt$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
'PRINT "PORT"; port, t$
IF LEFT$(t$, ibcnt%) <> "#B11111111" THEN GOSUB rderror
```

```
NEXT port
' LOCATE 20, 10: PRINT "Verify all PORTS and BITS HIGH"
IF noerror = 0 THEN LOCATE 20, 10: PRINT "RESET TEST PASSED"
IF noerror = 1 THEN LOCATE 20, 10: PRINT "RESET TEST FAILED"
GOSUB pauseroutine
CLS
t = 0
RETURN
'
OUTPUTBIT:
KEY OFF
CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
LOCATE 20, 10: PRINT "Running Alernating high low patern"
t$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
GOSUB findit
GOSUB basevalue
noerror = 0
E = 2
FOR port = 0 TO 11
wrt$ = "outportb(" + STR$(port) + ",#B10101010);"
wrt1$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibwrt(bd%, wrt1$)
CALL ibrd(bd%, t$)
IF LEFT$(t$, ibcnt%) <> "#B10101010" THEN GOSUB rderror
'PRINT "PORT"; port, LEFT$(t$, ibcnt%)
NEXT port
'LOCATE 20, 10: PRINT "Verify all PORTS alternate high low"
'GOSUB pauseroutine
CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
LOCATE 20, 10: PRINT "Running Alernating low high patern"
E = 3
FOR port = 0 TO 11
wrt$ = "outportb(" + STR$(port) + ",#B01010101);"
wrt1$ = "inportb(" + STR$(count) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibwrt(bd%, wrt1$)
CALL ibrd(bd%, t$)
IF LEFT$(t$, ibcnt%) <> "#B1010101" THEN GOSUB rderror
'PRINT "PORT"; count, LEFT$(t$, ibcnt%)
NEXT port
' LOCATE 20, 10: PRINT "Verify all PORTS alternate high low (7 bits)"
```

```
' GOSUB pauseroutine
 CLS
GOSUB resetbit
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
LOCATE 20, 10: PRINT "Running Walking One patern"
FOR port = 0 TO 11
RESTORE highbit
IF port >= 1 THEN
wrt3$ = "outportb(" + STR$(port - 1) + ",0);"
CALL ibwrt(bd%, wrt3$)
END IF
FOR bit = 0 TO 7
GOSUB wrtbit
GOSUB rdbit
GOSUB rdports
NEXT bit
NEXT port
CLS
GOSUB setbit
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
LOCATE 20, 10: PRINT "Running Walking Zero patern"
FOR port = 0 TO 11
RESTORE lowbit
IF port >= 1 THEN
wrt3$ = "outportb(" + STR$(port - 1) + ",0);"
CALL ibwrt(bd%, wrt3$)
END IF
FOR bit = 0 TO 7
GOSUB wrtbit1
GOSUB rdbit1
GOSUB rdport1
NEXT bit
NEXT port
CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
IF noerror = 0 THEN PRINT "OUTPUT BIT TEST PASSED"
IF noerror = 1 THEN PRINT "OUT[UT BIT TEST FAILED"
GOSUB pauseroutine
CLS
t = 0
RETURN
'
pauseroutine:
 ' pause routine for subroutines
```

```
 LOCATE 22, 1: PRINT "press any key to continue"
 C$ = INKEY$: WHILE C$ = "": C$ = INKEY$: WEND
RETURN
findit:
 Udname$ = "VXITECH"
 CALL IBFIND(Udname$, bd%)
 IF bd% < 0 THEN
 CLS
 LOCATE 10, 10: PRINT "Unable to locate or communicat with the VT-1"
 LOCATE 11, 10: PRINT "Check power and setup"
 STOP
 END IF
 RETURN
findit1:
 Udname$ = "ctrl0"
 CALL IBFIND(Udname$, bd%)
 IF bd% < 0 THEN
 CLS
 LOCATE 10, 10: PRINT "Unable to locate or communicat with the Slot 0"
 LOCATE 11, 10: PRINT "Check power and setup"
 STOP
 END IF
 RETURN
basevalue:
 wrt$ = "base(2);"
 CALL ibwrt(bd%, wrt$)
 RETURN
setbit:
 FOR V = 0 TO 11
 wrt$ = "outportb(" + STR$(V) + ",#hFF);"
 CALL ibwrt(bd%, wrt$)
 NEXT V
 RETURN
resetbit:
 FOR V = 0 TO 11
 wrt$ = "outportb(" + STR$(V) + ",0);"
 CALL ibwrt(bd%, wrt$)
 NEXT V
 RETURN
wrtbit:
wrt$ = "setbit(" + STR$(port) + "," + STR$(bit) + ");"
CALL ibwrt(bd%, wrt$)
RETURN
wrtbit1:
```

```
wrt$ = "rstbit(" + STR$(port) + "," + STR$(bit) + ");"
CALL ibwrt(bd%, wrt$)
RETURN
rdbit:
wrt$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
READ A$
IF LEFT$(t$, ibcnt%) <> A$ THEN
noerror = 1
LOCATE 12, 10: PRINT "Walking One Bit Test ERROR"
LOCATE 14, 10: PRINT "Port"; port, "IS: "; LEFT$(t$, ibcnt%), "S/B: "; A$
GOSUB pauseroutine
END IF
 CLS
 LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
 LOCATE 20, 10: PRINT "Running Walking One patern"
RETURN
rdbit1:
wrt$ = "inportb(" + STR$(port) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
READ A$
IF LEFT$(t$, ibcnt%) <> A$ THEN
noerror = 1
LOCATE 12, 10: PRINT "Walking Zero Bit Test ERROR"
LOCATE 14, 10: PRINT "Port"; port, "IS: "; LEFT$(t$, ibcnt%), "S/B: "; A$
GOSUB pauseroutine
END IF
  CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
 LOCATE 20, 10: PRINT "Running Walking Zero patern"
RETURN
rdports:
IF port = 11 THEN RETURN
FOR readports = port + 1 TO 11
wrt$ = "inportb(" + STR$(readports) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
IF LEFT$(t$, ibcnt%) <> "#B0" THEN
noerror = 1
LOCATE 12, 10: PRINT "Walking One Bit Test ERROR"
LOCATE 14, 10: PRINT "Sent Port "; port; " Value "; LEFT$(t$, ibcnt%)
LOCATE 15, 10: PRINT "Failure in Port "; readports; " S/B #B0"
```

```
GOSUB pauseroutine
END IF
 NEXT readports
CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
LOCATE 20, 10: PRINT "Running Walking One patern"
RETURN
rdport1:
IF port = 11 THEN RETURN
FOR readports = port + 1 TO 11
wrt$ = "inportb(" + STR$(readports) + ");"
CALL ibwrt(bd%, wrt$)
CALL ibrd(bd%, t$)
IF LEFT$(t$, ibcnt%) <> "#B11111111" THEN
noerror = 1
LOCATE 12, 10: PRINT "Walking Zero Bit Test ERROR"
LOCATE 14, 10: PRINT "Sent Port "; port; " Value "; LEFT$(t$, ibcnt%)
LOCATE 15, 10: PRINT "Failure in Port "; readports; " S/B #B0"
GOSUB pauseroutine
END IF
  CLS
LOCATE 2, 26: PRINT "OUTPUT BIT TEST"
 LOCATE 20, 10: PRINT "Running Walking Zero patern"
 NEXT readports
RETURN
rderror:
 IF E = 0 THEN sb$ = "#B1111111"
 IF E = 1 THEN sb$ = "#B0"
 IF E = 2 THEN sb$ = "#B10101010"
 IF E = 3 THEN sb$ = "#B1010101"
 CLS
 LOCATE 10, 10: PRINT "ERROR FOUND:  IS: PORT"; port; LEFT$(t$, ibcnt%), "S/B: "; sb$
 noerror = 1
 GOSUB pauseroutine
 RETURN
crderror:
 LOCATE 10, 10: PRINT "Card Address S/B: "; digits, "Card Address is: "; STR$(card)
 noerror = 1
 GOSUB pauseroutine
 RETURN
highbit:
DATA #B1
DATA #B11
DATA #B111
```

```
DATA #B1111
DATA #B11111
DATA #B111111
DATA #B1111111
DATA #B11111111
lowbit:
DATA #B11111110
DATA #B11111100
DATA #B11111000
DATA #B11110000
DATA #B11100000
DATA #B11000000
DATA #B10000000
DATA #B0

STOPPER:
 KEY OFF
 CLS
 END
```

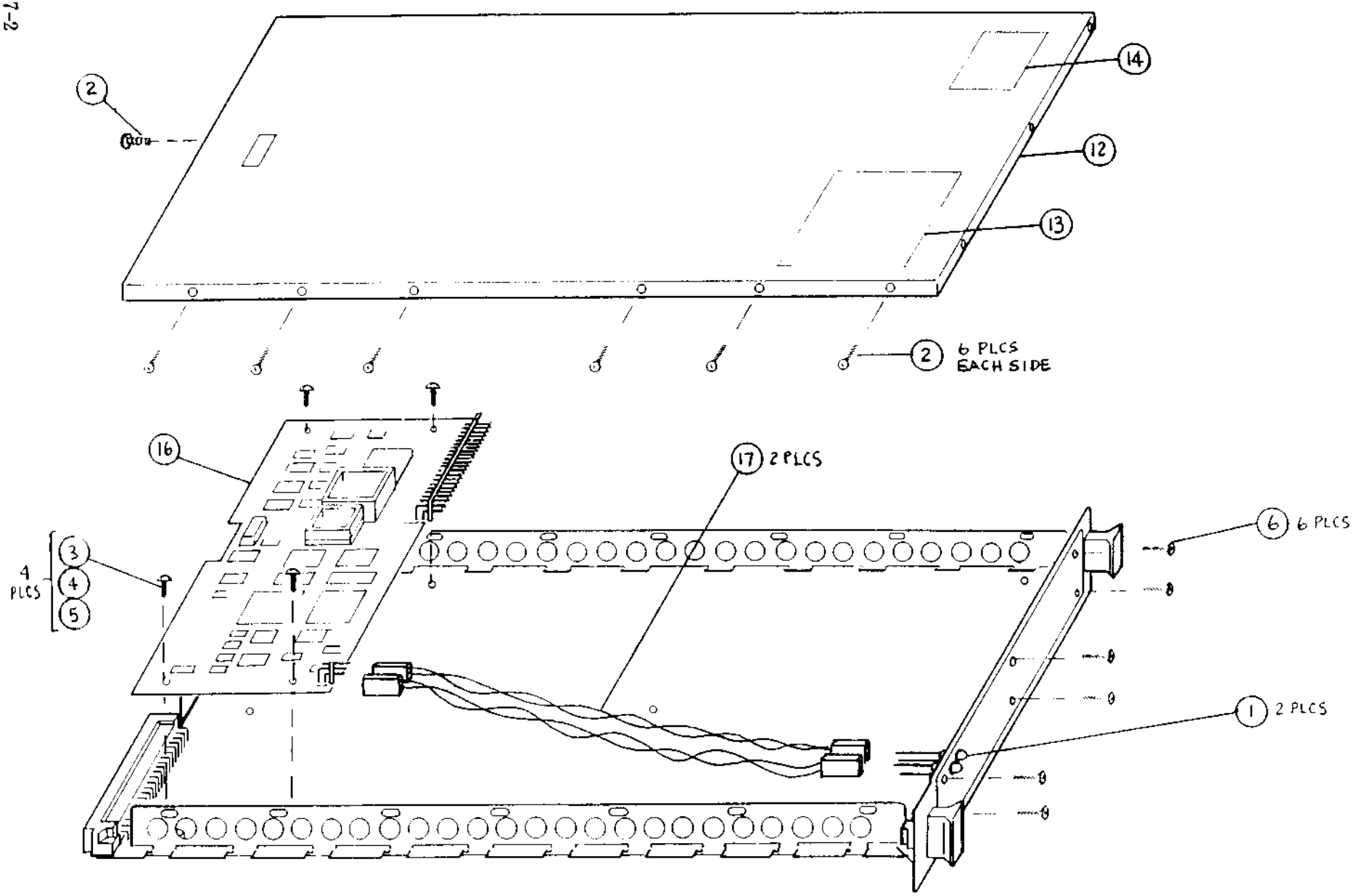
SECTION 7 DRAWINGS

Model 7064

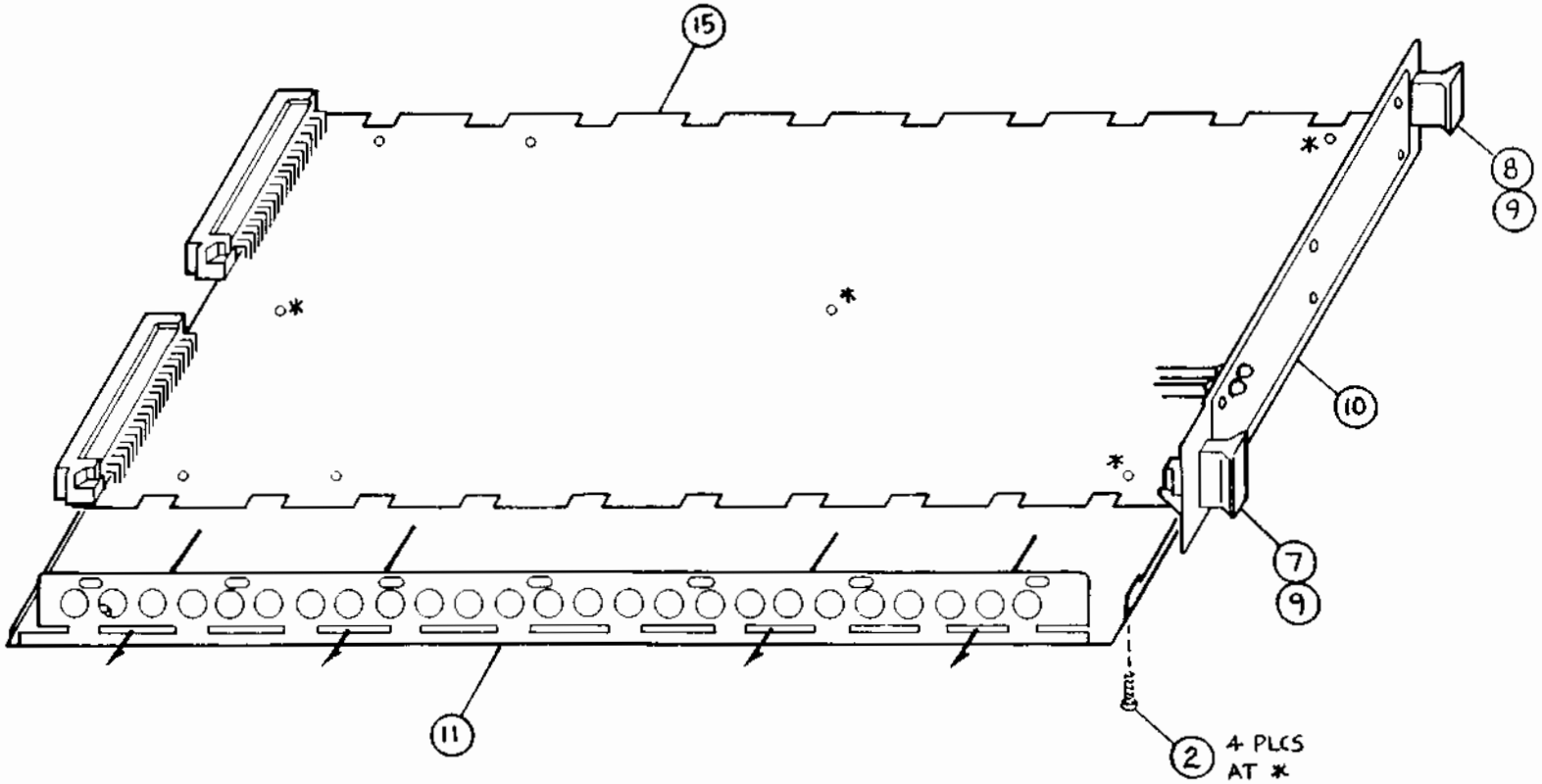
70-0010-000	Final Assembly	7-2
52-0010-000	PCB Assembly, "C" Size Breadboard	7-4
50-0010-000	Schematic, "C" Size Breadboard	7-6
52-0011-000	PCB Assembly, Message-Based Interface	7-17
50-0011-000	Schematic, Message-Based Interface	7-19
	Dimensional Drawings/Breadboard Hole Pattern	7-28

Model 7064-20

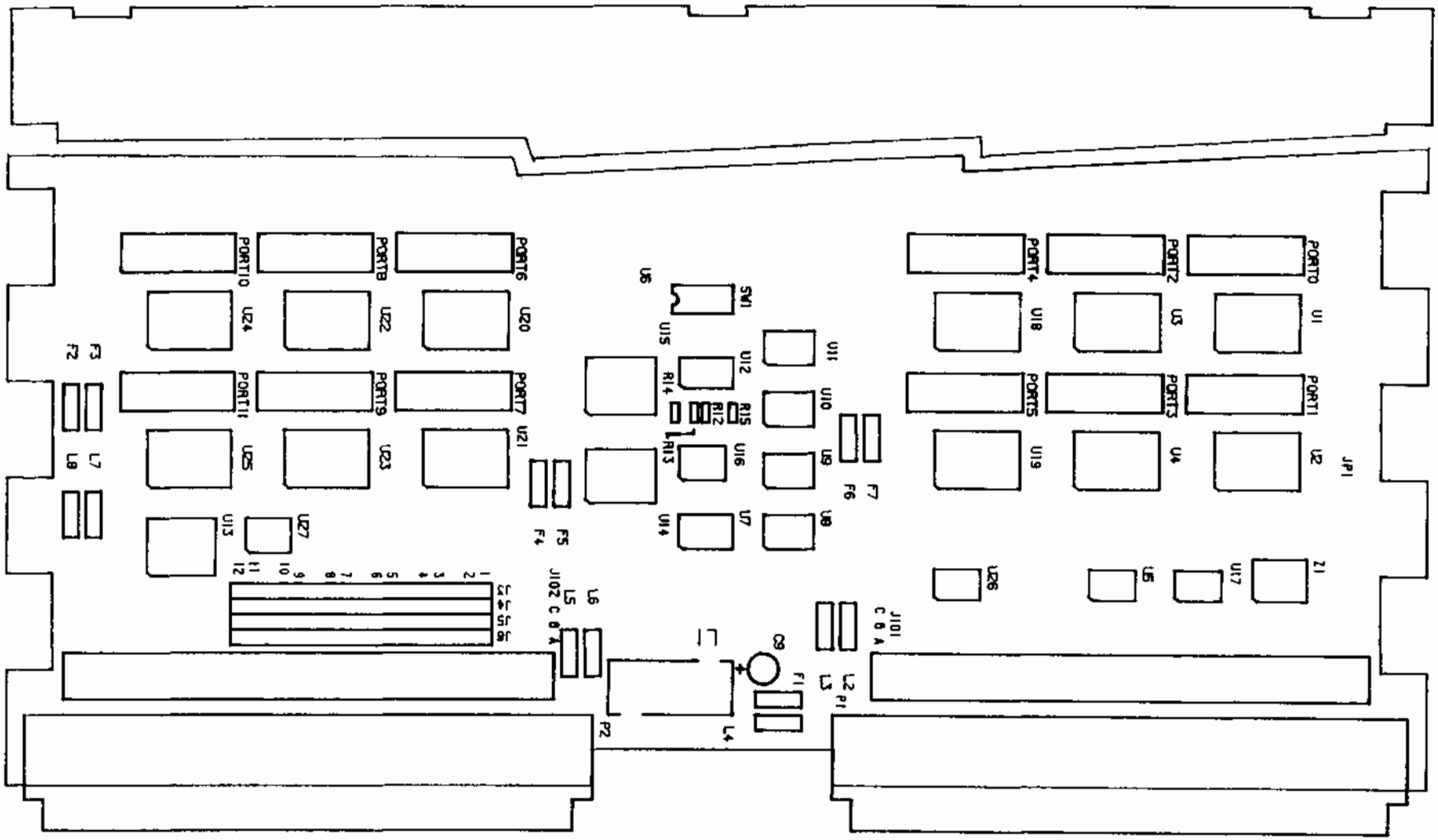
52-0026-000	PCB Assembly, Register-Based Breadboard	7-31
50-0026-000	Schematic, Register-Based Breadboard	7-33
	Dimensional Drawing/Breadboard Hole Pattern	7-52



FINAL ASSY., 7064			
SIZE B	CODE IDENT NO 21793	DWG NO 70-0010-000	REV 1
SCALE —		SHEET 1 OF 2	



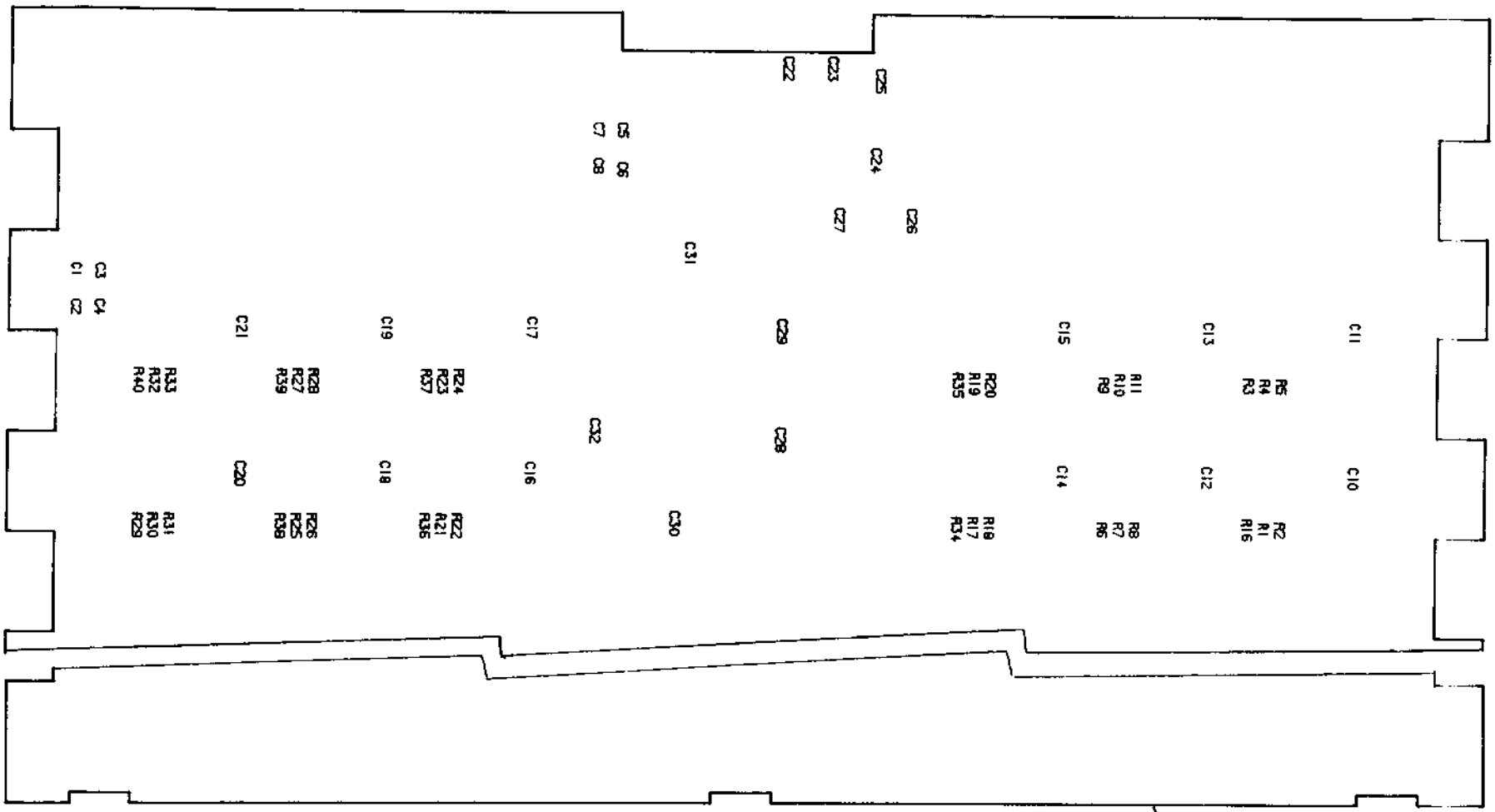
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	70-0010-000	1
SCALE —		SHEET 2 OF 2	



TOP SIDE

PCB ASSY., "C" SIZE BREADBOARD			
A	21793	52-0010-000	1

NOTES: SEE SHEET 2

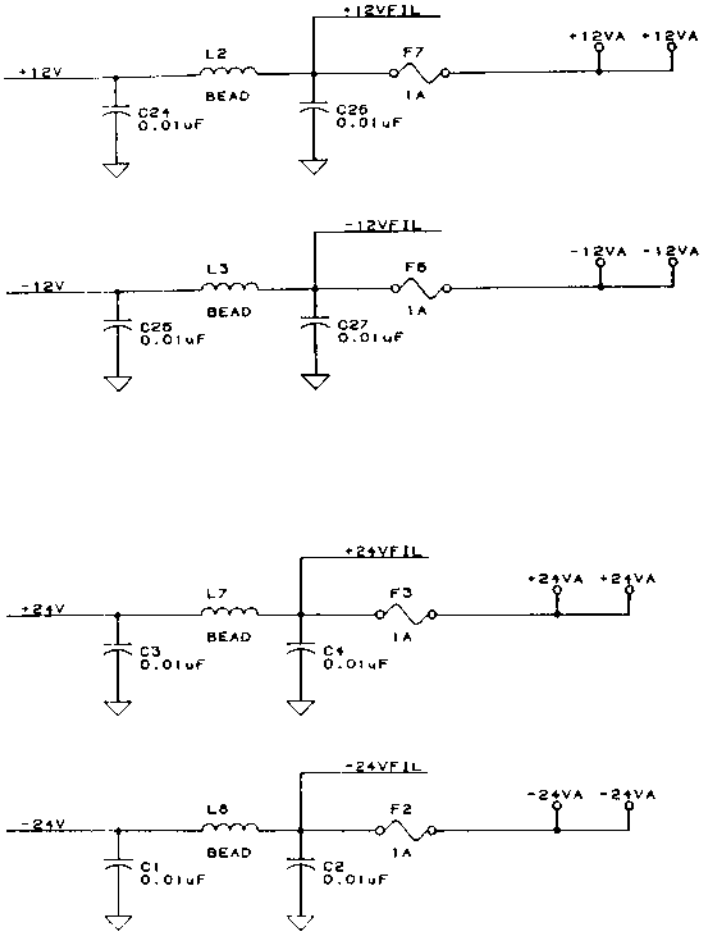


6. INSTALL WIRE JUMPERS BETWEEN J4-1 AND J5-1 THROUGH J4-10 AND J5-10
5. INSTALL WIRE JUMPERS BETWEEN PINS 1 AND 2 ON PORTS 0 THROUGH 11
4. INSTALL J101A,B,C AND J102A,B,C ON THE TOP SIDE OF PCB
3. INSTALL P1 AND P2 ON THE TOP SIDE OF PCB
2. INSTALL SW1 ON THE BOTTOM SIDE OF PCB
1. ASSEMBLY TO CONFORM TO VTI WORKMANSHIP STANDARDS

NOTES: UNLESS OTHERWISE SPECIFIED:

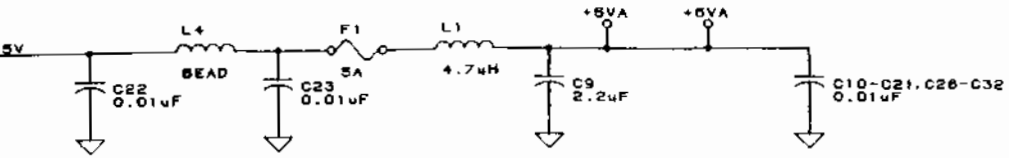
SIZE	FSCM NO	DWG NO	REV
A	21793	52-0010-000	1

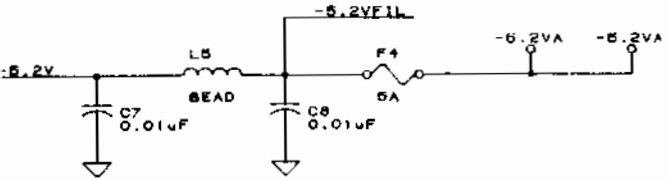
SCALE — SHEET 2 OF 2

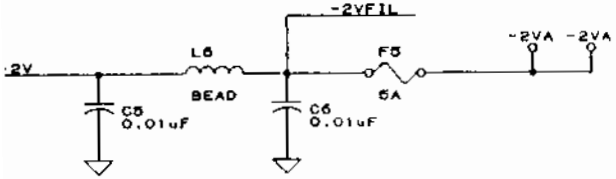


3. INSTALL WIRE JUMPER BETWEEN PIN 1 AND PIN 2 ON PORT0 THROUGH PORT11 CONNECTORS
2. ALL CAPACITORS ARE 20% 50V
1. ALL RESISTORS ARE 5% 1/8 WATT

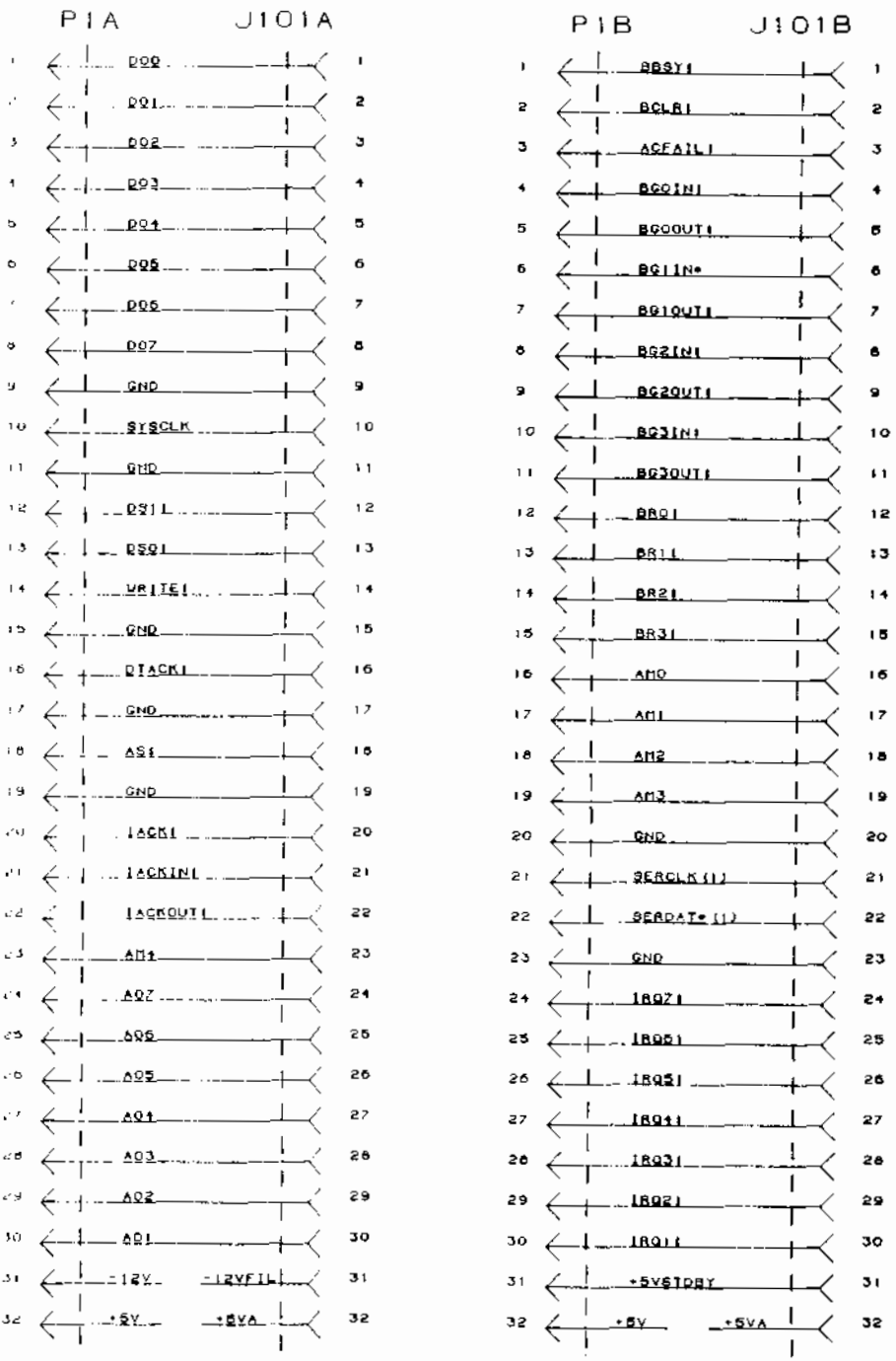
NOTES: UNLESS OTHERWISE SPECIFIED

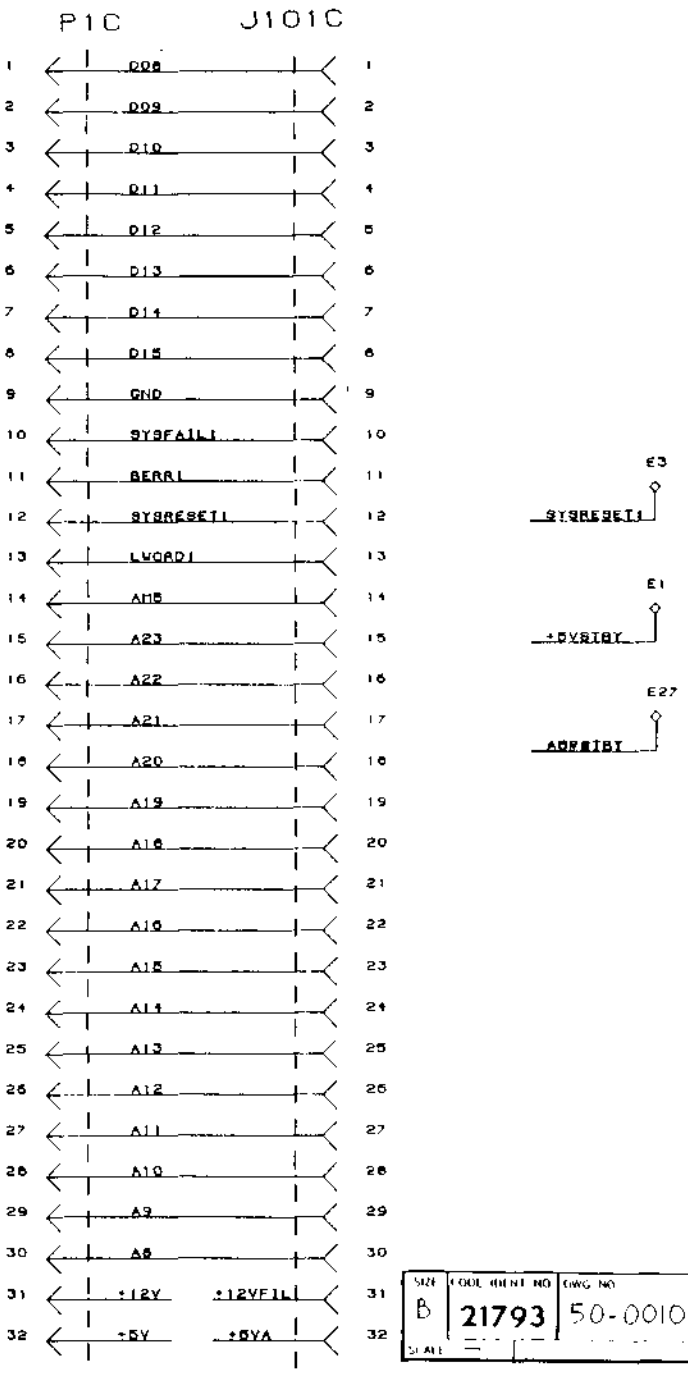




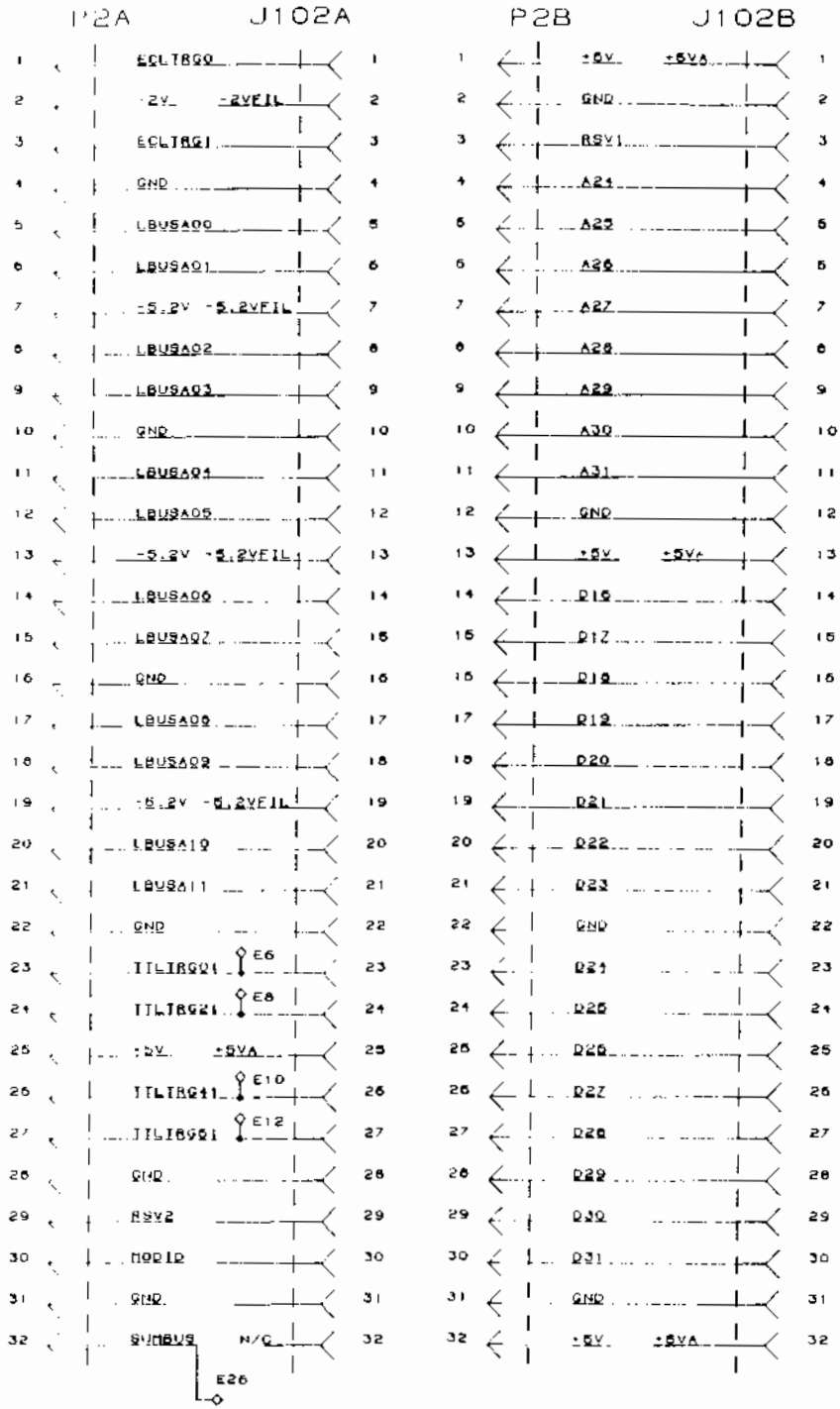


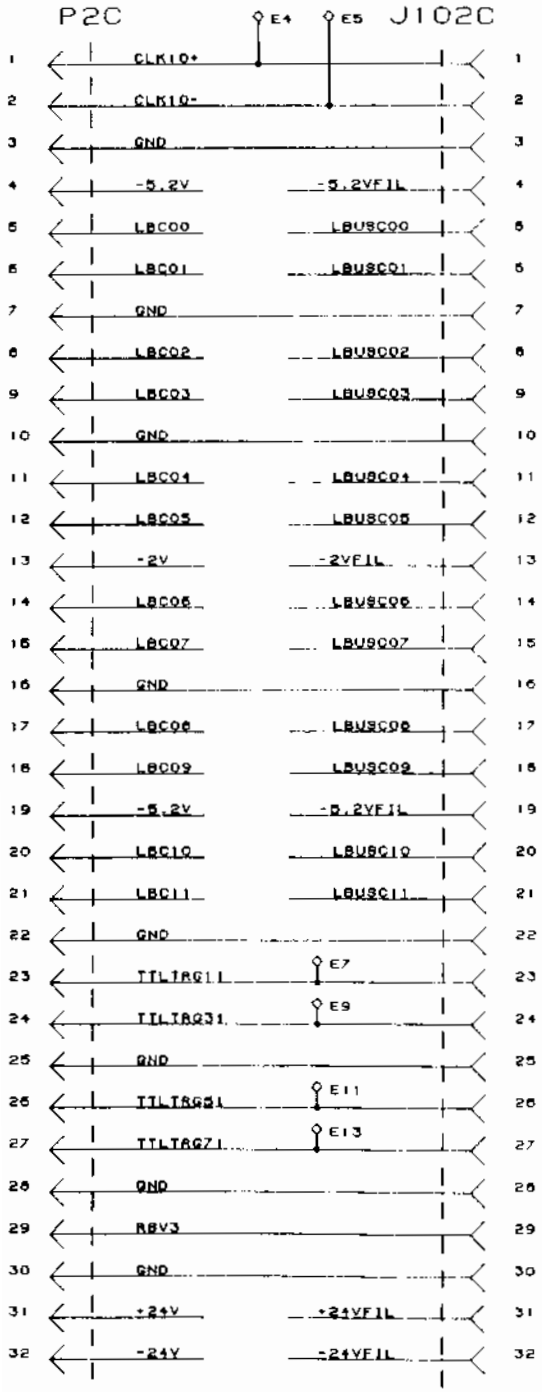
SCHEMATIC, VXI "C" SIZE BREADBOARD			
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 1 OF 11	


P1A
J101A
D00
D01
D02
D03
D04
D05
D06
D07
GND
SYSCLK
GND
DS1L
DS0L
WRITE
GND
DTACK
GND
AS
GND
IACK
IACKIN
IACKOUT
AM1
A07
A06
A05
A04
A03
A02
A01
-12V
-12VFIL
+5V
+5VA
P1B
J101B
BBSY
BCLR
ACFAIL
BG0IN
BG0OUT
BG1IN*
BG1OUT
BG2IN
BG2OUT
BG3IN
BG3OUT
BR0
BR1
BR2
BR3
AM0
AM1
AM2
AM3
GND
SERCLK(1)
SERDAT*(1)
GND
IRQ7
IRQ6
IRQ5
IRQ4
IRQ3
IRQ2
IRQ1
+5VSTDBY
+5V
+5VA



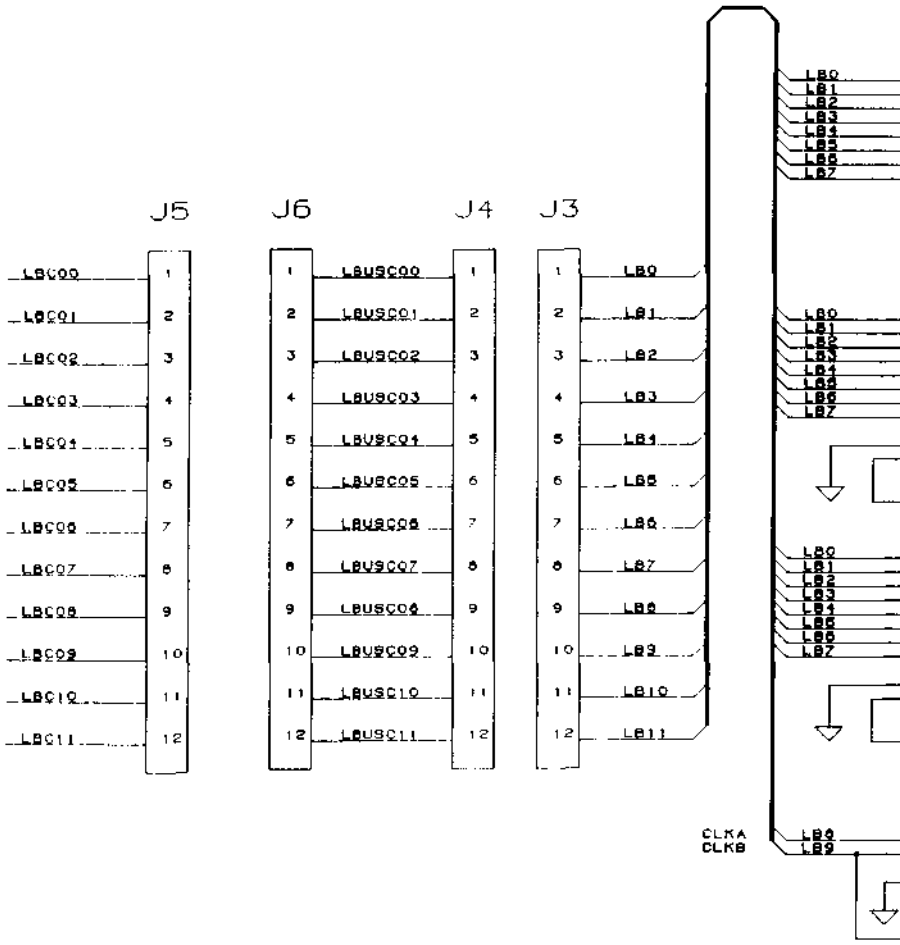
SIZE	TOOL IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE		SHEET 2 OF 11	

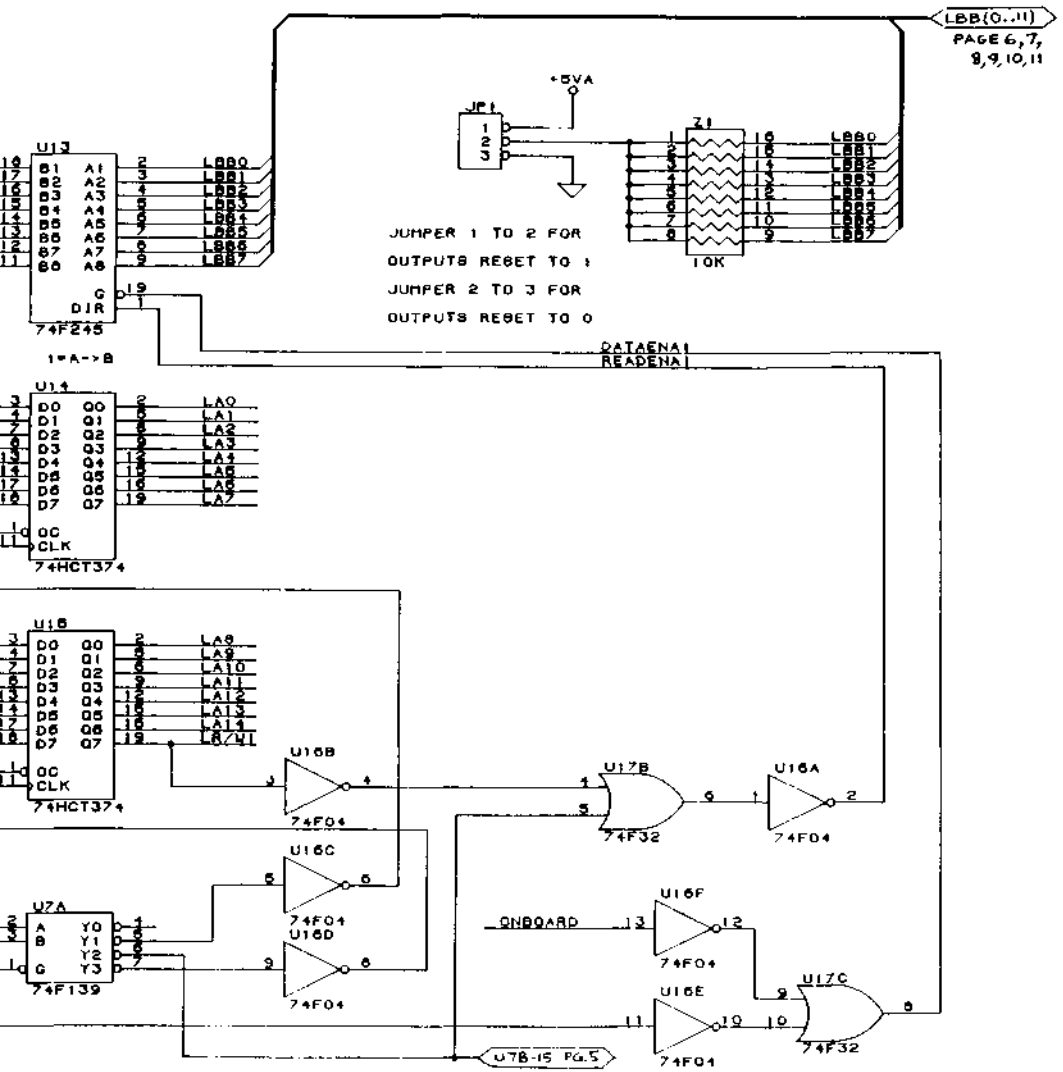

P2A
J102A
P2B
J102B
ECLTBG0
-2V
-2VFIL
ECLTBG1
GND
LBUSA00
LBUSA01
-5.2V
-5.2VFIL
LBUSA02
LBUSA03
GND
LBUSA04
LBUSA05
-5.2V
-5.2VFIL
LBUSA06
LBUSA07
GND
LBUSA08
LBUSA09
-5.2V
-5.2VFIL
LBUSA10
LBUSA11
GND
TTLTBG0
TTLTBG1
+5V
+5VA
TTLTBG4
TTLTBG5
GND
RSV2
MODID
GND
SUMBUS
N/C
E6
E8
E10
E12
E26
+5V
+5VA
RSV1
A24
A25
A26
A27
A28
A29
A30
A31
GND
+5V
+5VA
D16
D17
D18
D19
D20
D21
D22
D23
GND
D24
D25
D26
D27
D28
D29
D30
D31
GND
+5V
+5VA



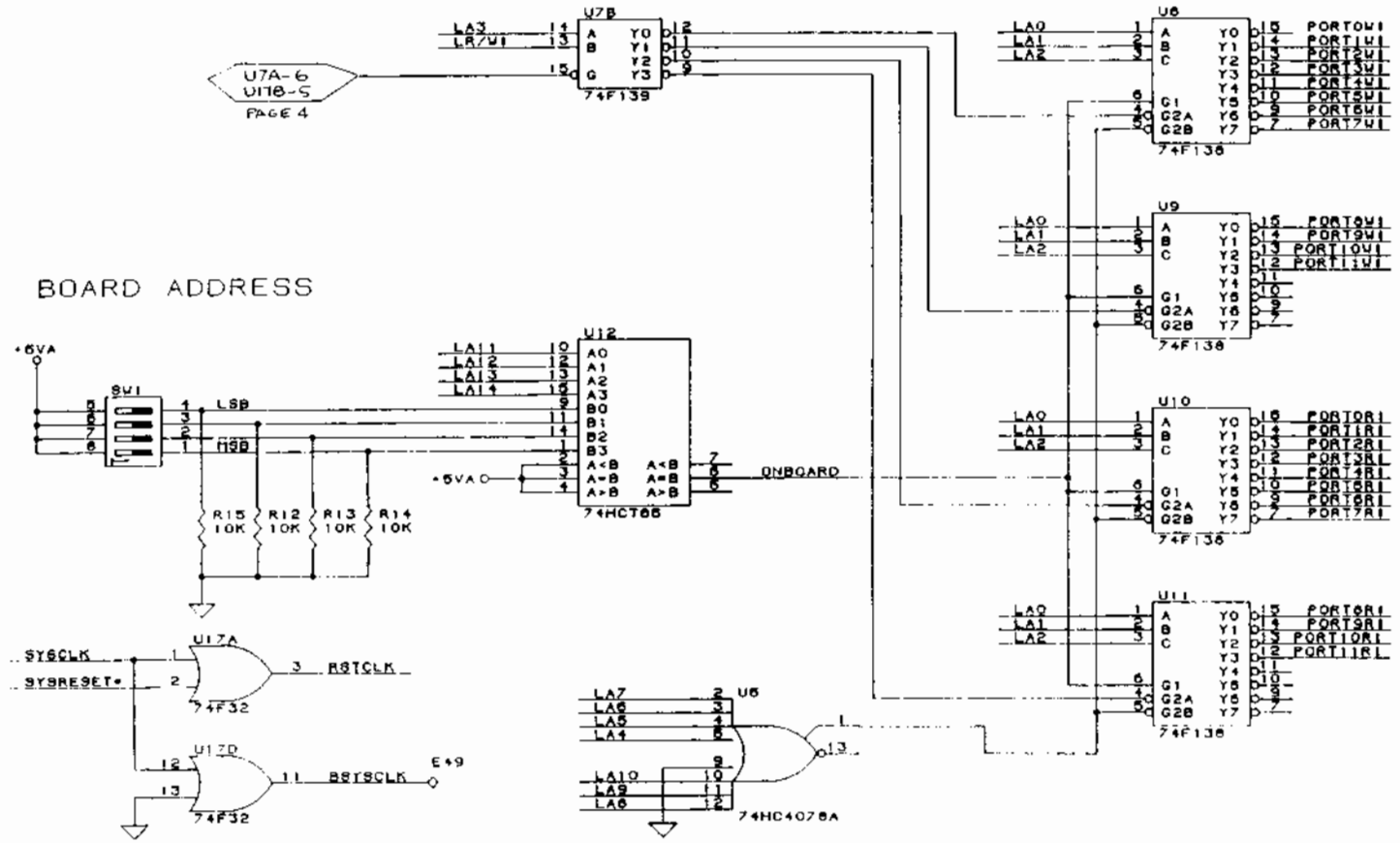
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 3 OF 11	

[redacted]

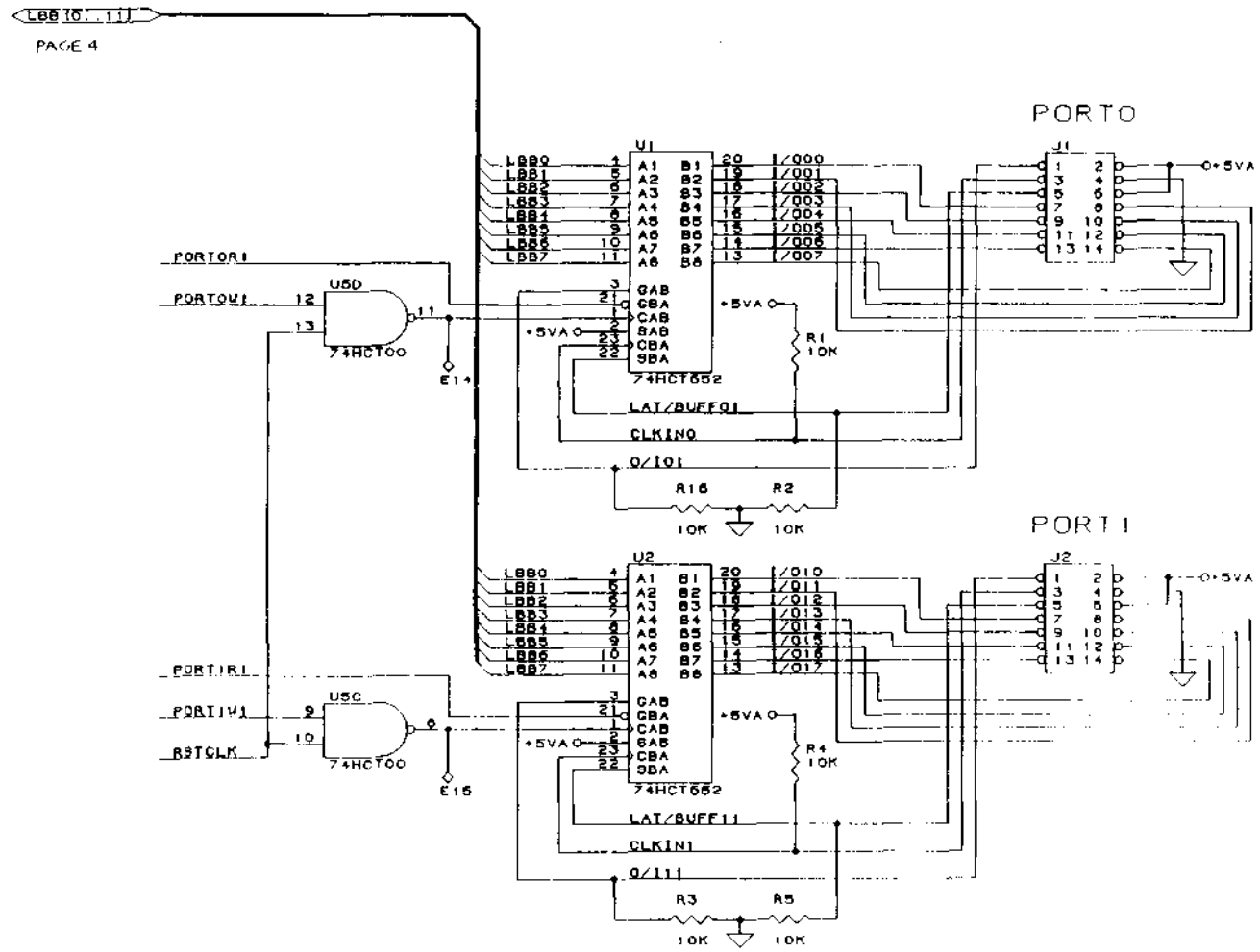

J5
J6
J4
J3
LBC00
LBC01
LBC02
LBC03
LBC04
LBC05
LBC06
LBC07
LBC08
LBC09
LBC10
LBC11
LBUSC00
LBUSC01
LBUSC02
LBUSC03
LBUSC04
LBUSC05
LBUSC06
LBUSC07
LBUSC08
LBUSC09
LBUSC10
LBUSC11
LB0
LB1
LB2
LB3
LB4
LB5
LB6
LB7
LB8
LB9
LB10
LB11
CLKA
CLKB



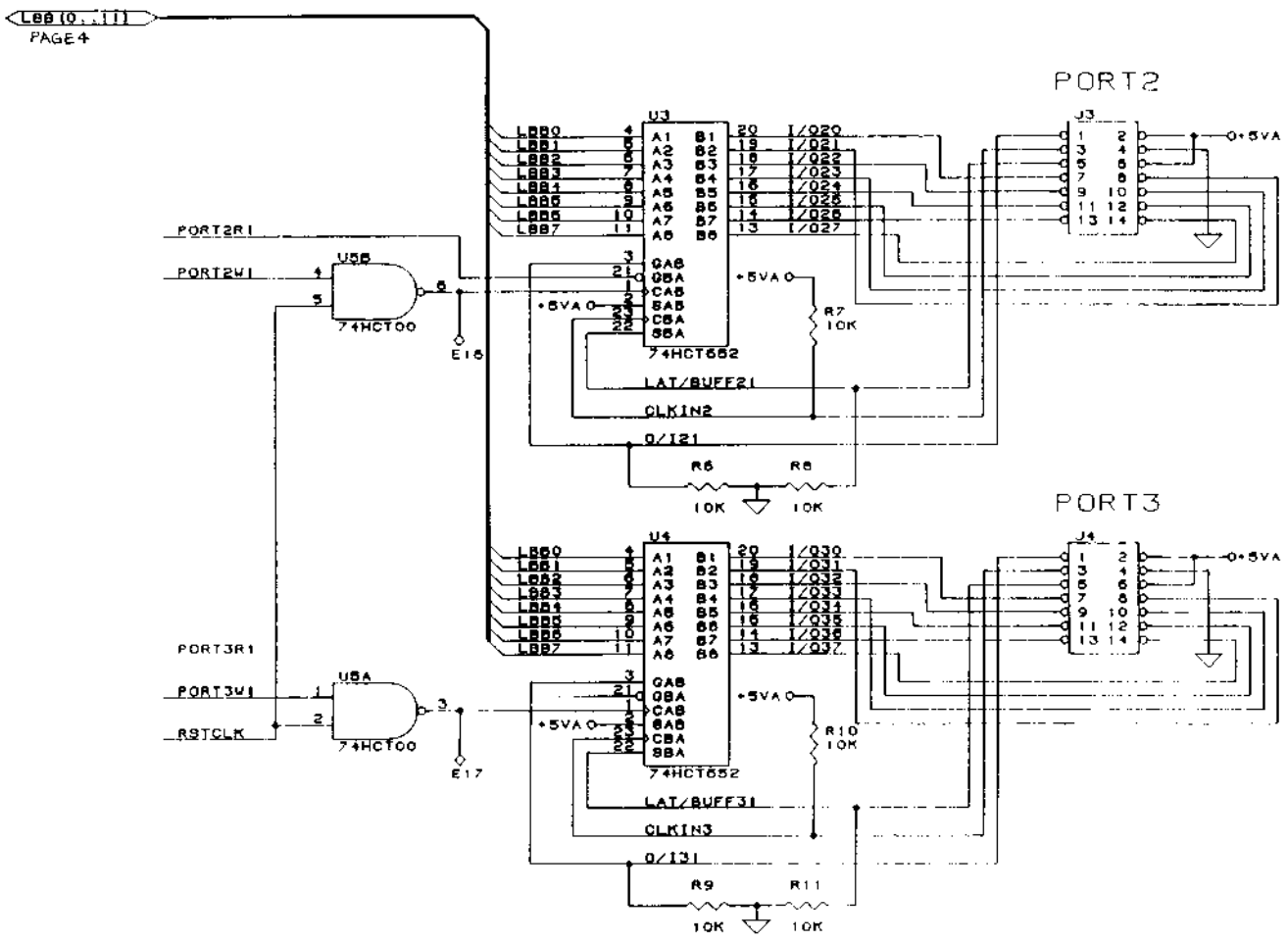
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE		SHEET 4 OF 11	



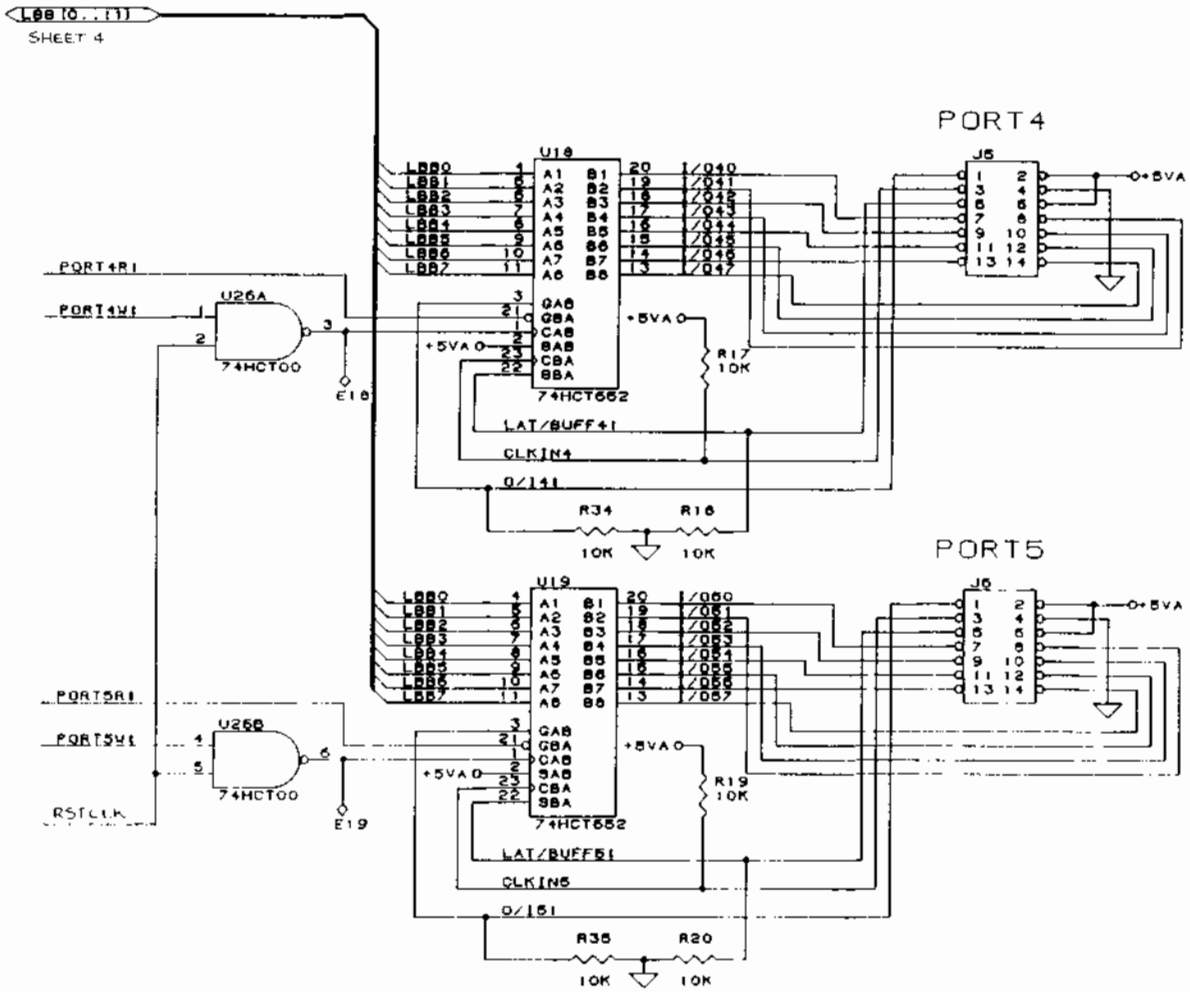
SIZE	FILE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE		SHEET 5 OF 11	



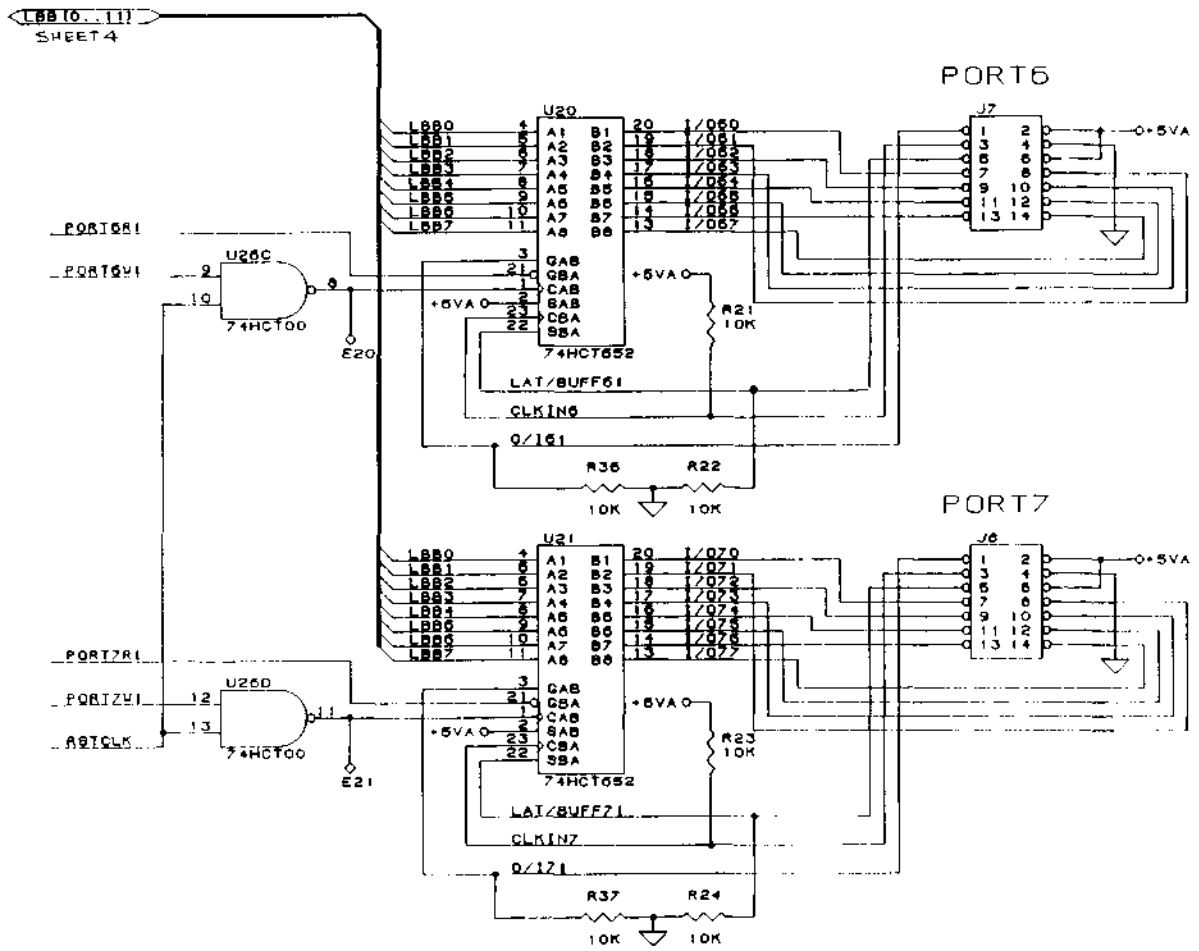
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 6 OF 11	



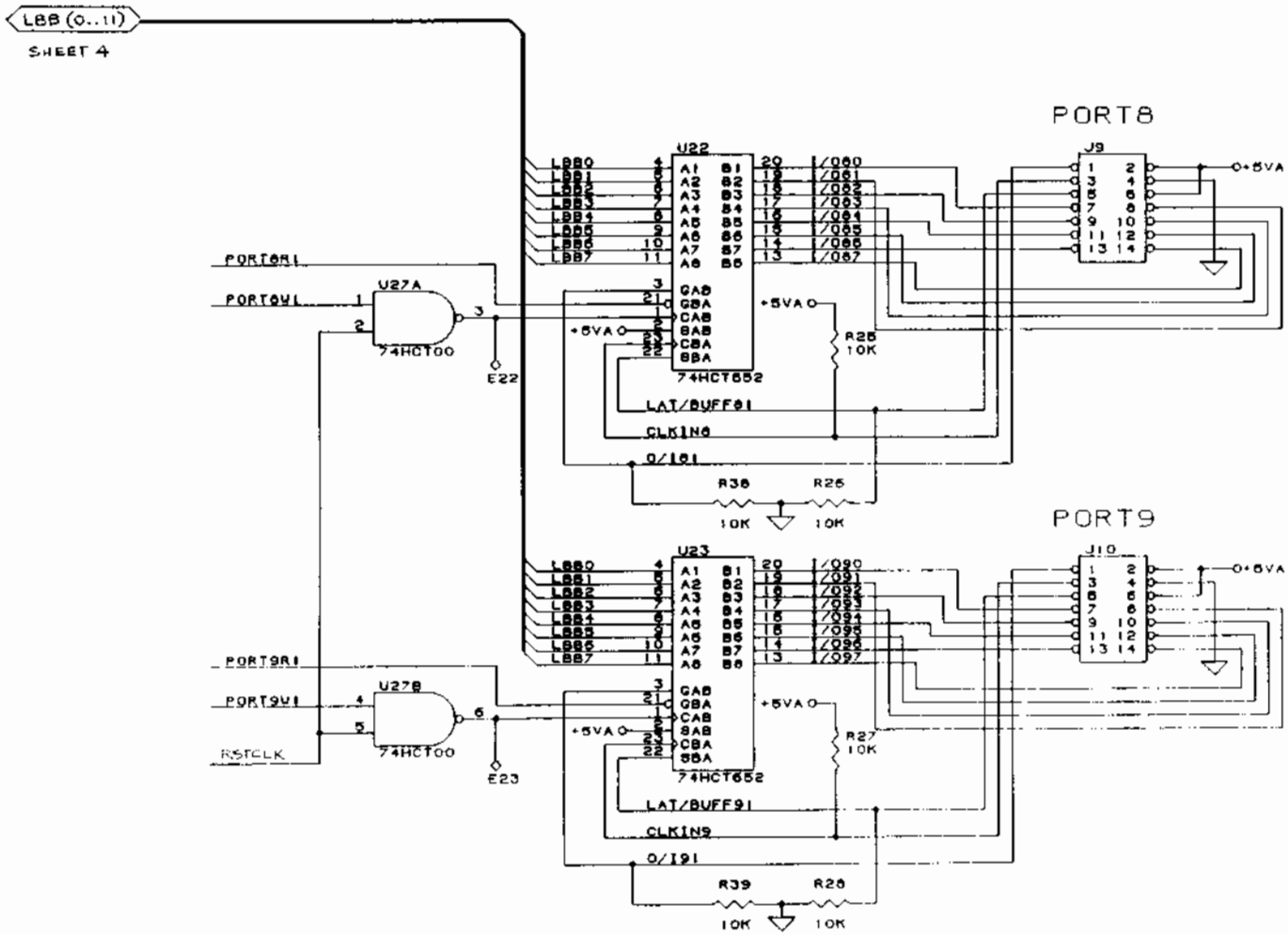
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
		SHEET 7 OF 11	



SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 8 OF 11	

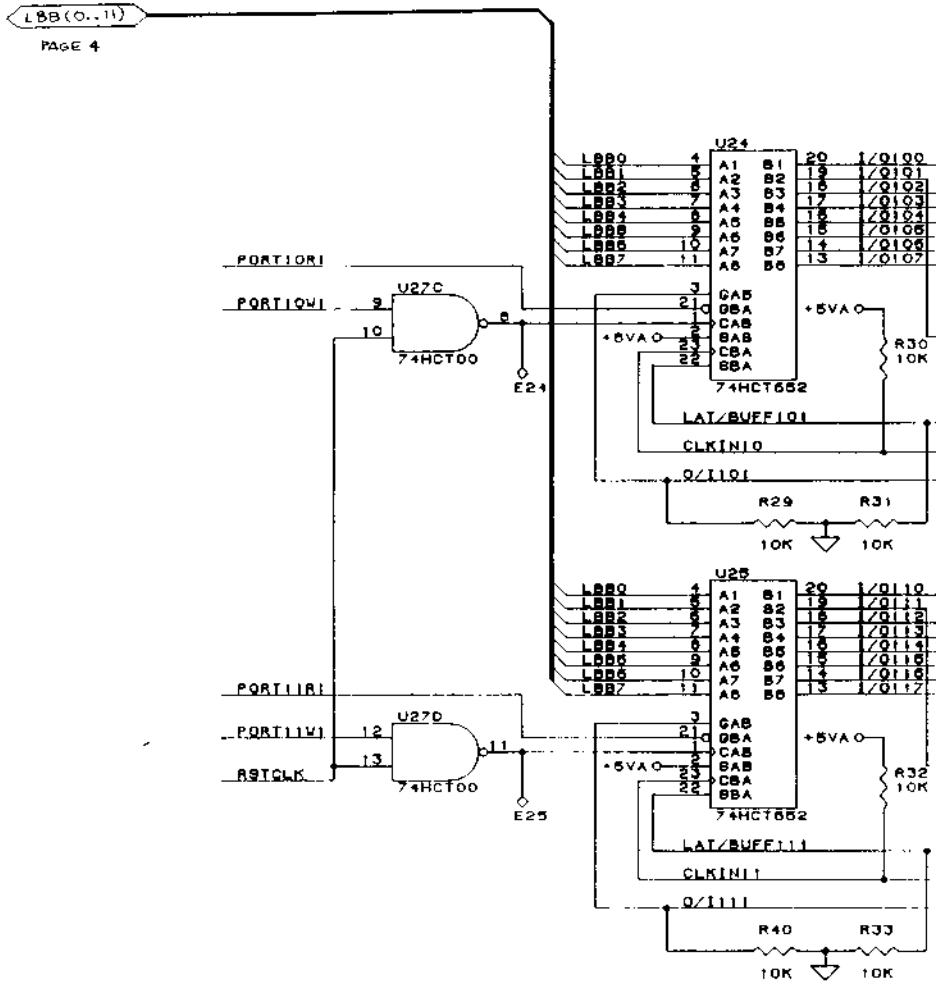


SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 9 OF 11	

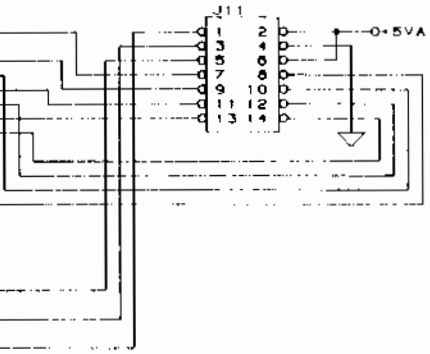


SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A

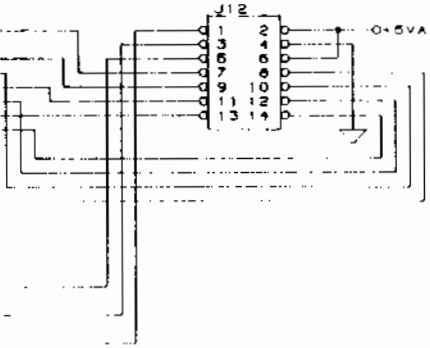
SCALE — SHEET 10 OF 11


LBB(0..11)
PAGE 4
U24
U25
U27C
U27D
74HCT00
74HCT652
PORT10RI
PORT10WI
PORT11RI
PORT11WI
RSTCLK
E24
E25
+5VA
R30
10K
R31
R32
R33
R29
R40
LAT/BUFF10
CLKIN10
O/I10
LAT/BUFF11
CLKIN11
O/I11

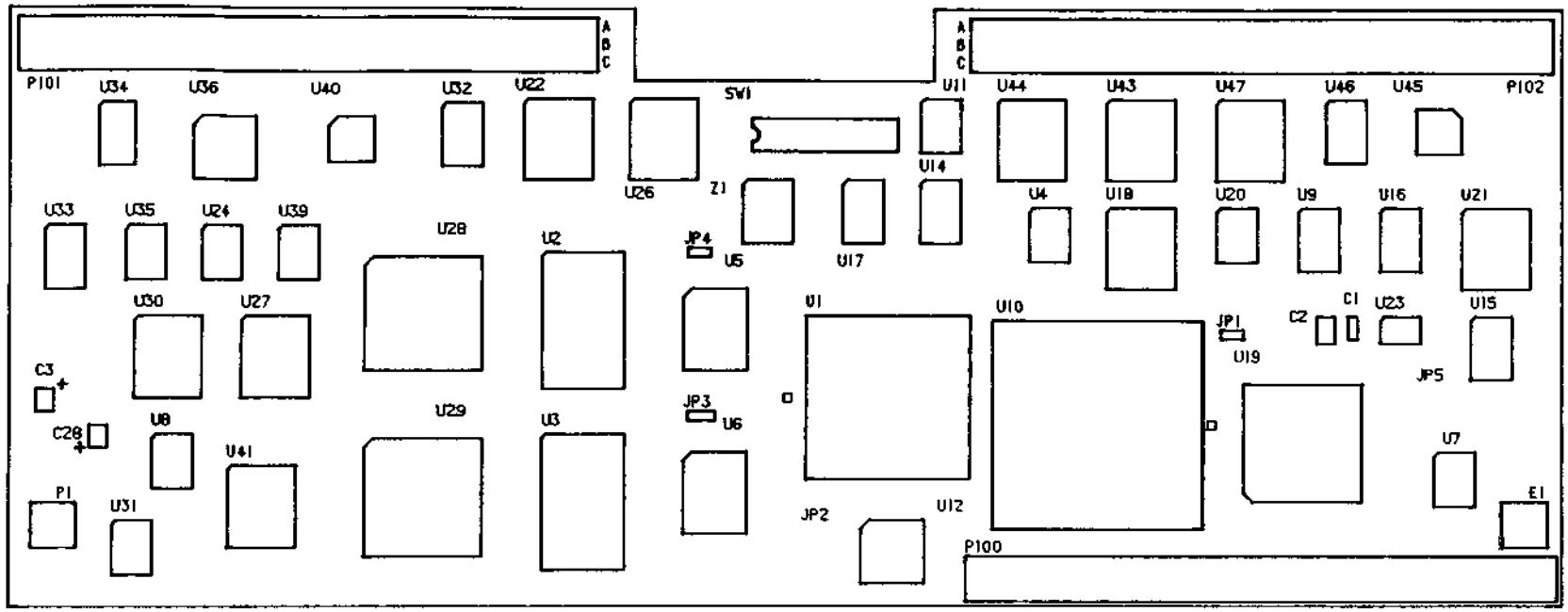
PORT10



PORT11

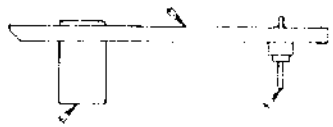


SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0010-000	A
SCALE —		SHEET 11 OF 11	



TOP SIDE

TOP SIDE OF PCB



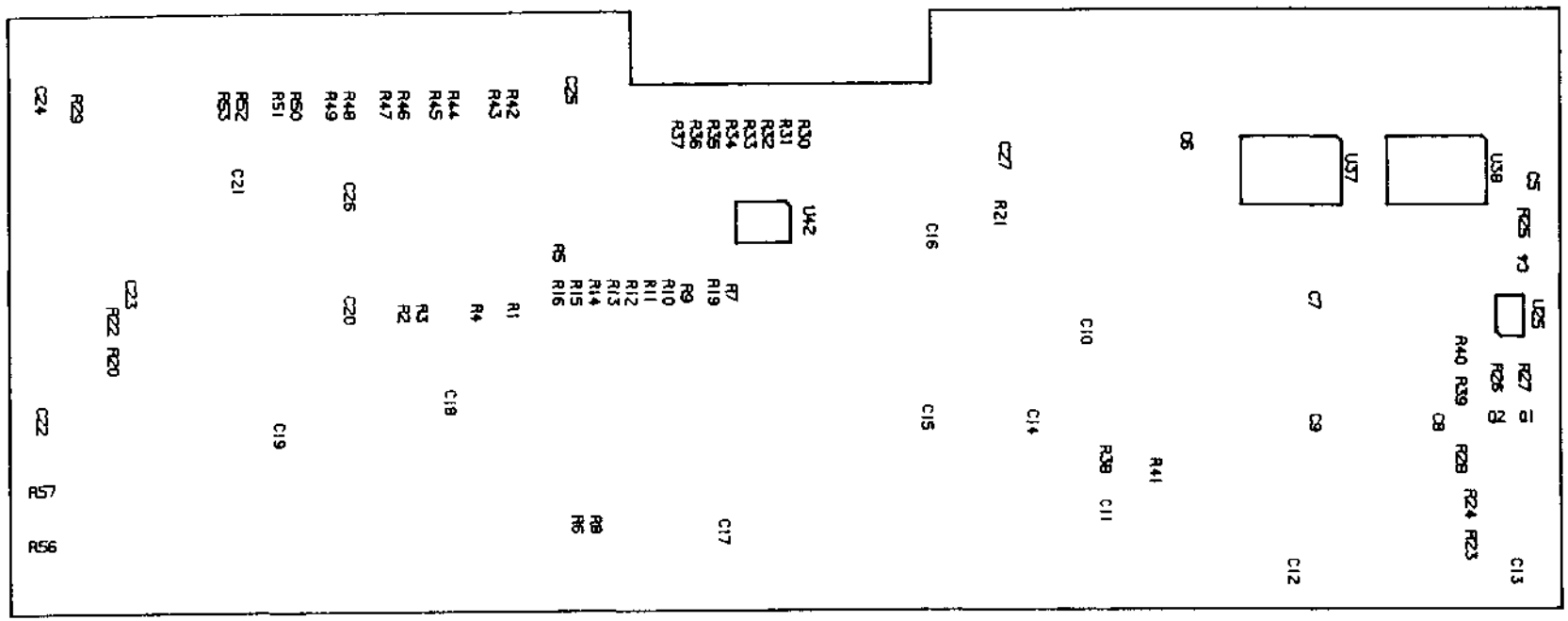
INSTALL AND SWAGE AS SHOWN

INSTALL PINS AS SHOWN
AT P101A,B,C AND P102A,C

3. P1 AND P100 ARE INSTALLED ON THE TOP SIDE OF PCB
2. REFER TO 50-0011-000 FOR SCHEMATIC
1. ASSEMBLY IS TO CONFORM TO VTI WORKMANSHIP STANDARDS

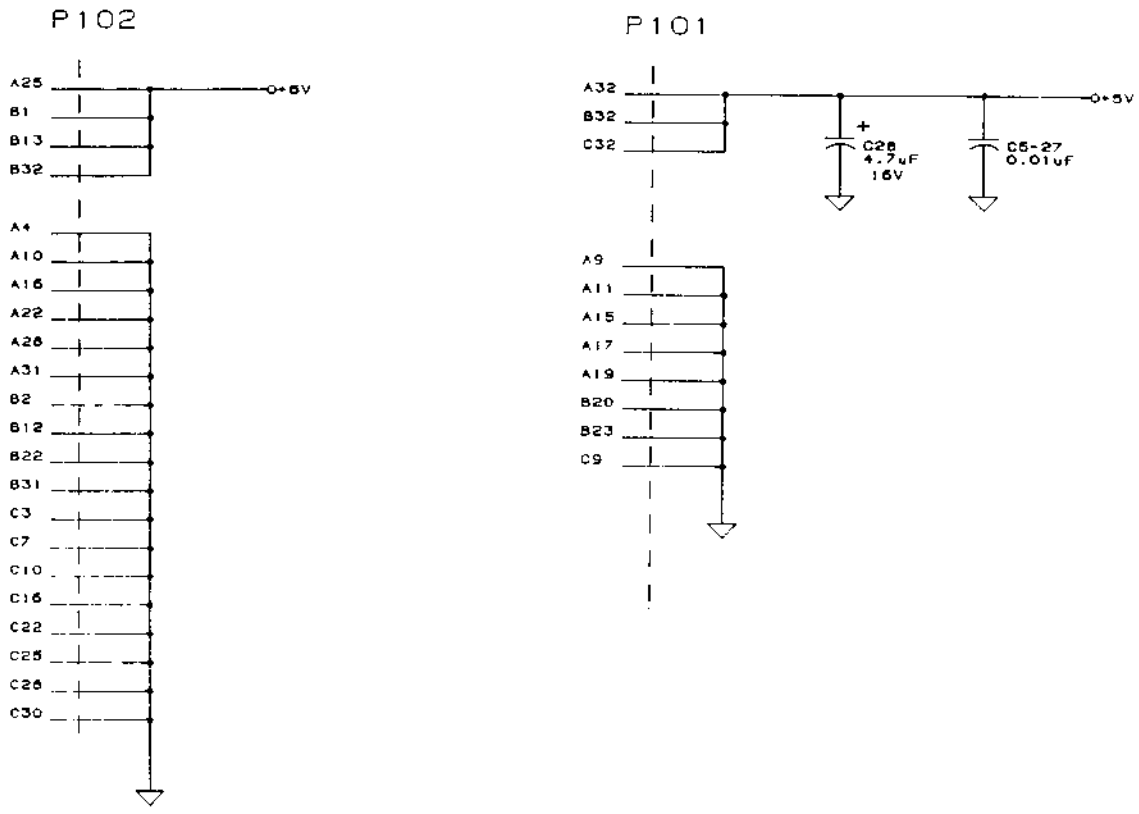
NOTES: UNLESS OTHERWISE SPECIFIED.

PCB ASSY., VXI MESSAGE BASED INT			
SIZE A	CAGE CODE 21793	52-0011-000	SHEET 1 OF 2



BOTTOM SIDE

A	21793	52-0011-000	1
			SHEET 2 OF 2



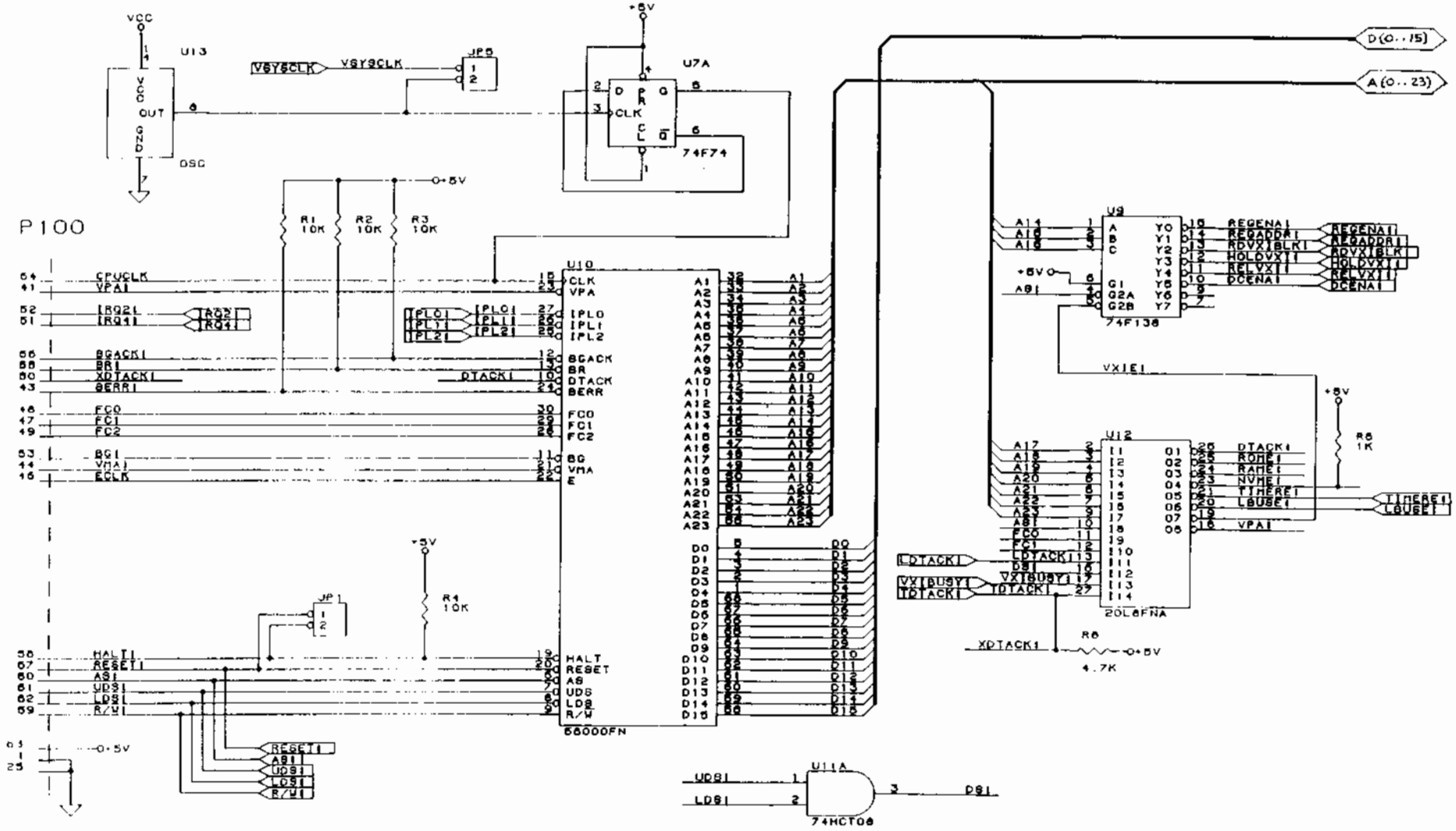
5. U13 AND JP5 ARE INSTALLED ON 12.5 MHz UNITS ONLY
4. INSTALL JP2 FOR 256K X 16 EPROM
   DELETE JP2 FOR 64K X 16 AND 128K X 16 EPROM
3. INSTALL JP3 AND JP4 FOR 32K X 8 NVM
   DELETE JP3 AND JP4 FOR 8K X 8 NVM
2. ALL CAPACITORS ARE 20% 50V
1. ALL RESISTORS ARE 5% 1/8 WATT

NOTES: UNLESS OTHERWISE SPECIFIED

SCHEMATIC, MESSAGE BASED DEVICE

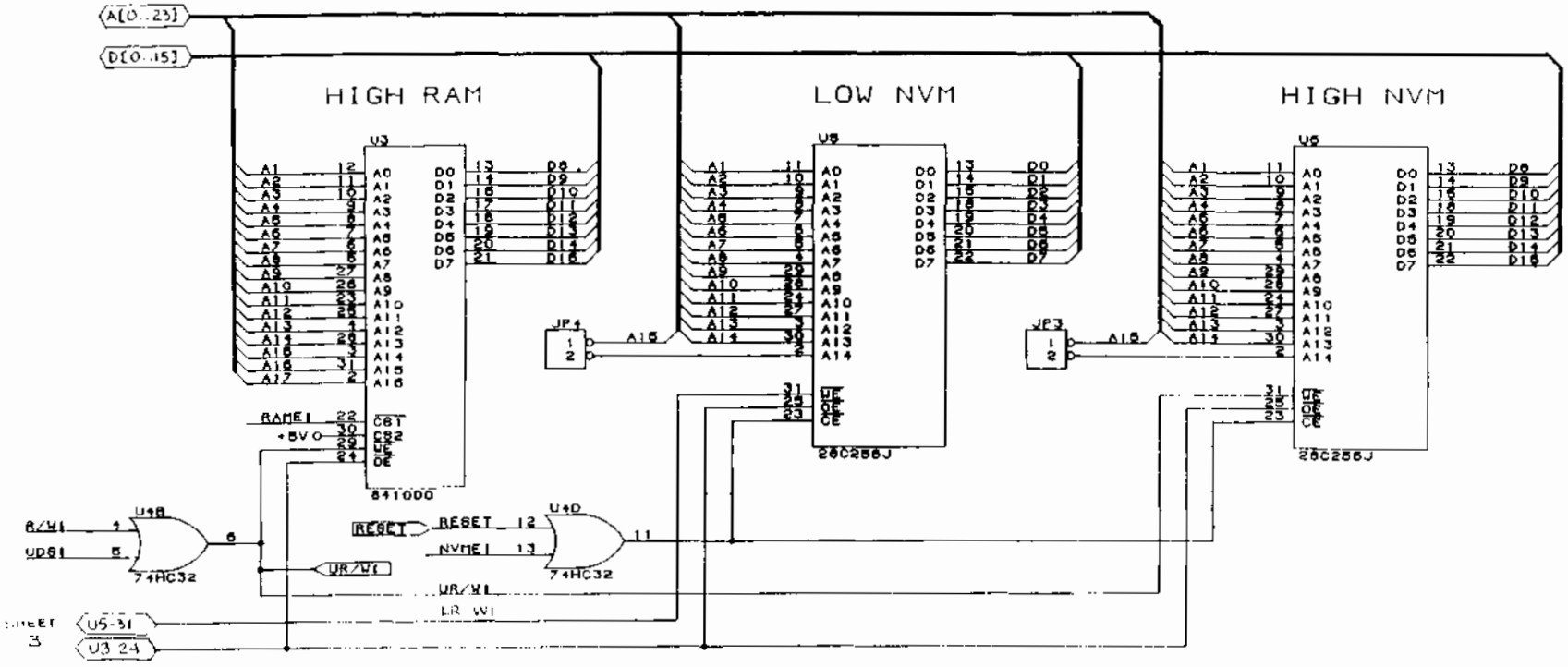
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0011-000	A

SCALE: NONE SHEET 1 OF 9

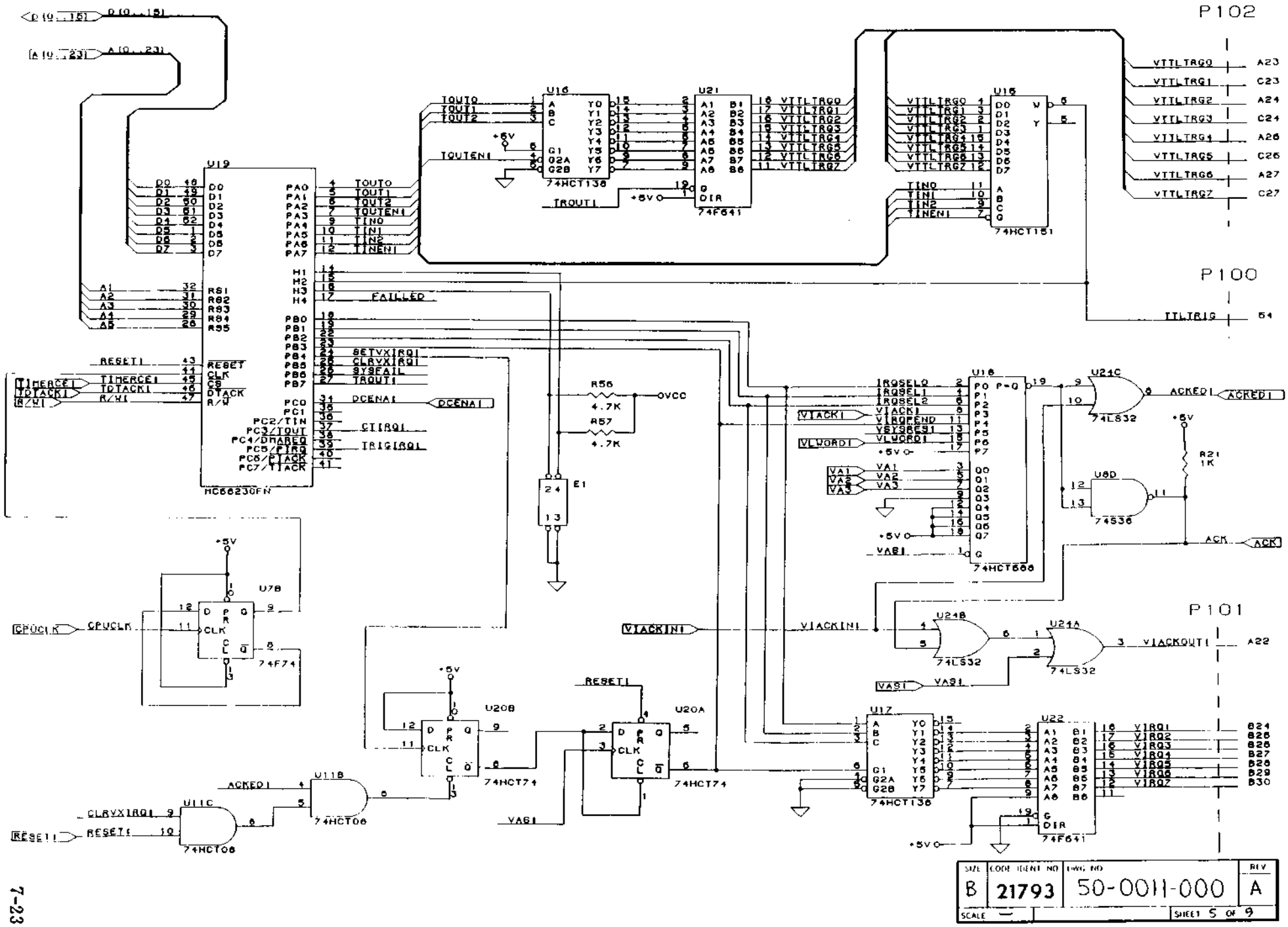


SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0011-000	A
SCALE —		SHEET 2 OF 9	

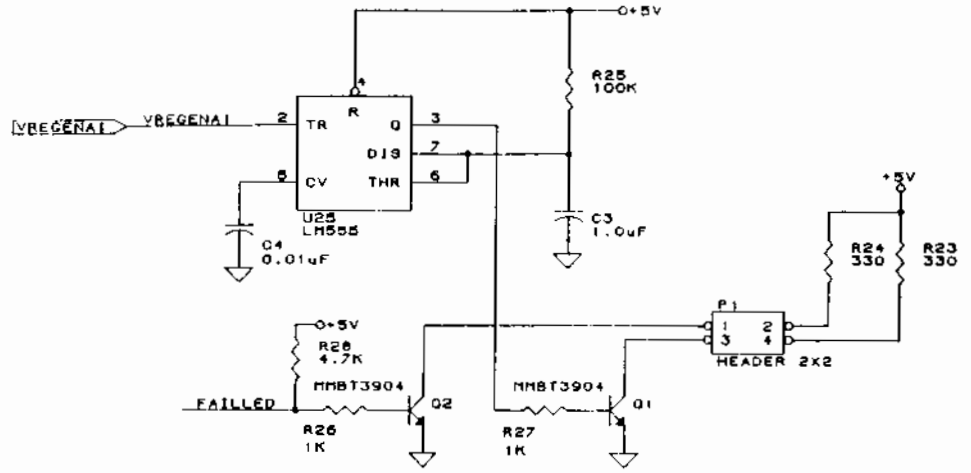
PROGRAM ROM
LOW RAM
P100
ROME 00000000h-0FFFFFFh DTACK
RAME 40000000h-5FFFFFFh DTACK
NVME 60000000h-6FFFFFFh DTACK
TIMERE 70000000h-71FFFFFh DTACK
LBUSE 78000000h-7BFFFFFh DTACK
VXIE 7C000000h-7FFFFFFh DTACK
REGENA 7D000000
REGADDR 7C40000
RDVXIBLK 7C80000
HOLDVXI 7CC0000
RELVXI 7D00000
A[0..23]
D[0..15]
JP2
A19
R7 4.7K
+5V
ROME
RAME
27C1024HG
74HC32
74S38
R/W
LDS
LR/W
SHEET 4
FOR 27C1024 OR 27C2048 DELETE R10
FOR 27C4096 INSTALL R10
SIZE B
FSCM NO 21793
DWG NO 50-0011-000
REV A
SHEET 3 OF 9

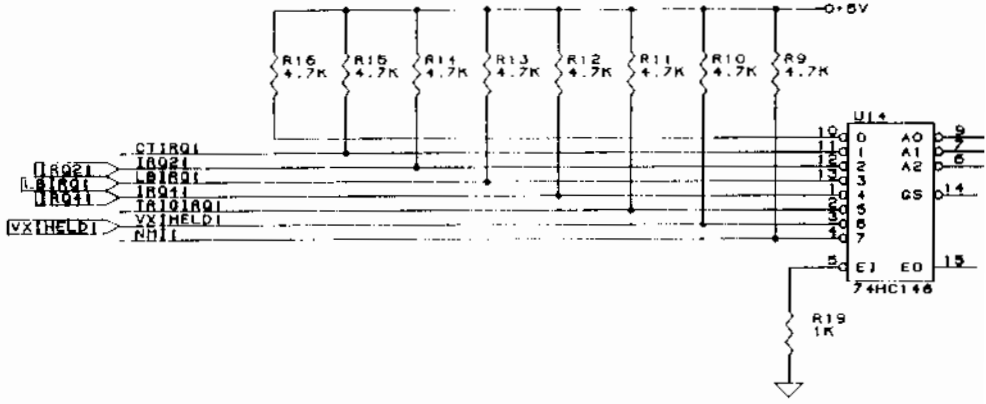


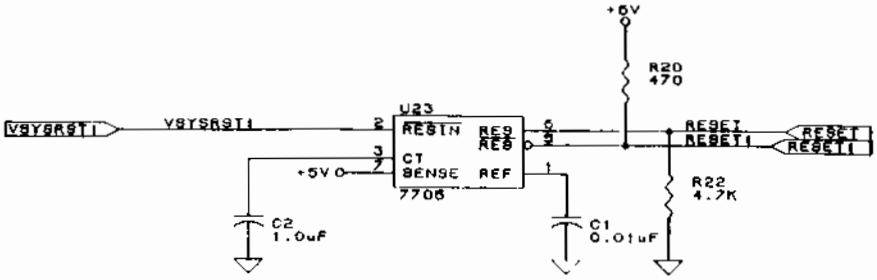
SIZE	FSCM IDENT NO	DWG NO	REV
B	21793	50-0011-000	A
SCALE —		SHEET 4 OF 9	

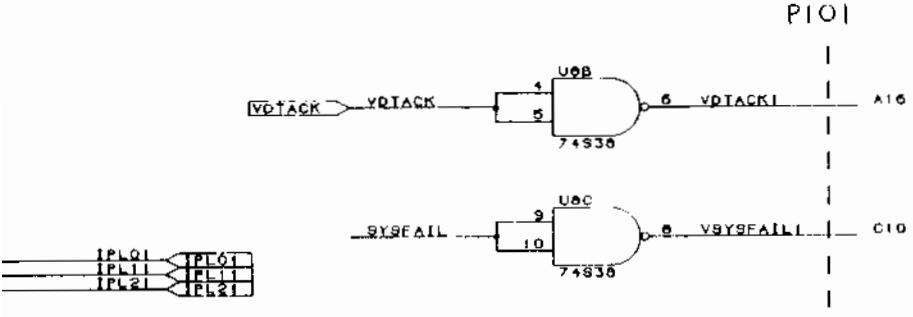


SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0011-000	A
SCALE —		SHEET 5 OF 9	

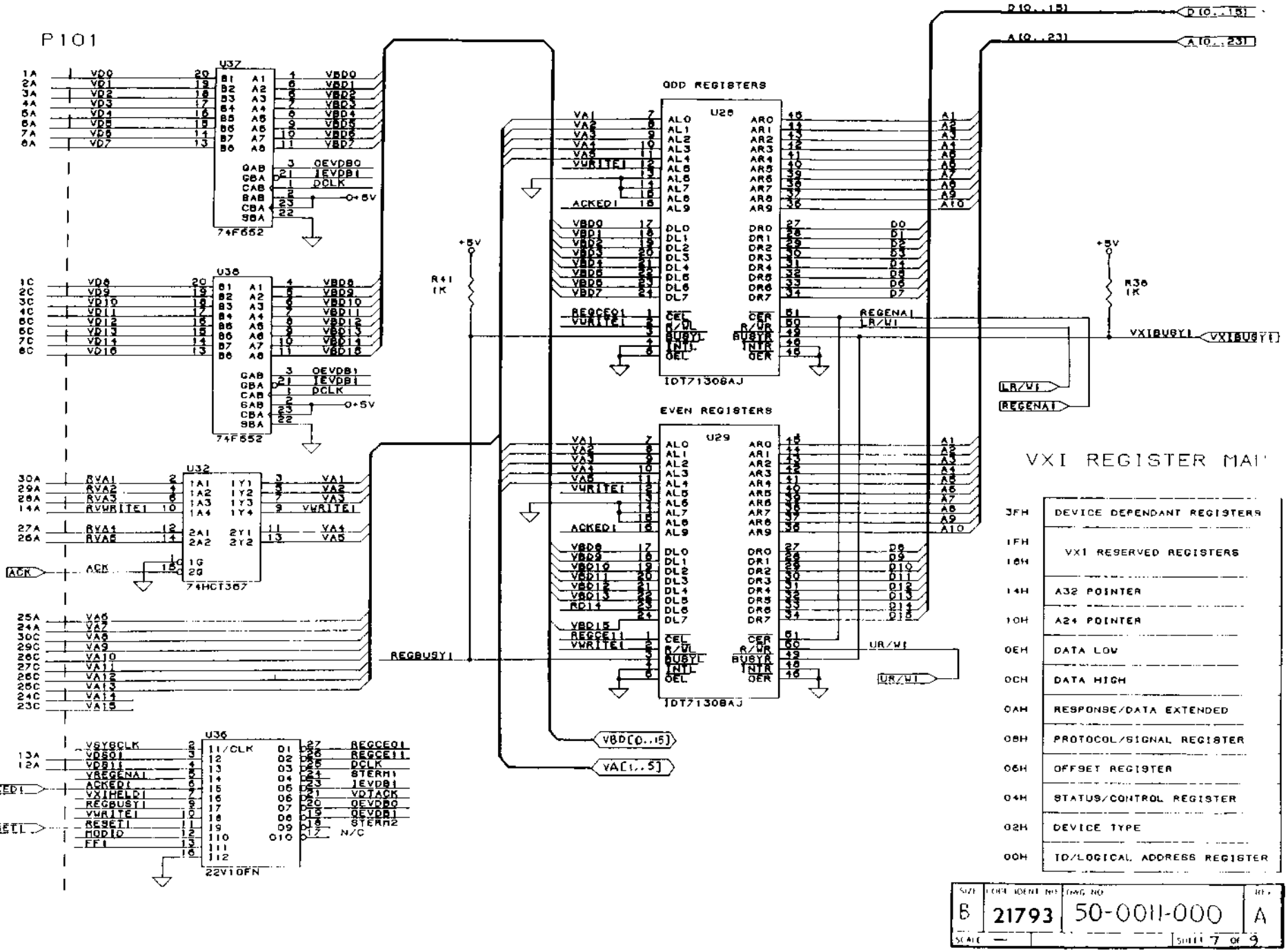

+5V
R25
100K
VBEGENAI
VBEGENAI
TR
R
Q
DIS
CV
THR
U25
LM555
C4
0.01uF
C3
1.0uF
+5V
R24
330
R23
330
P1
HEADER 2X2
+5V
R28
4.7K
MMBT3904
Q2
FAILLED
R26
1K
MMBT3904
Q1
R27
1K

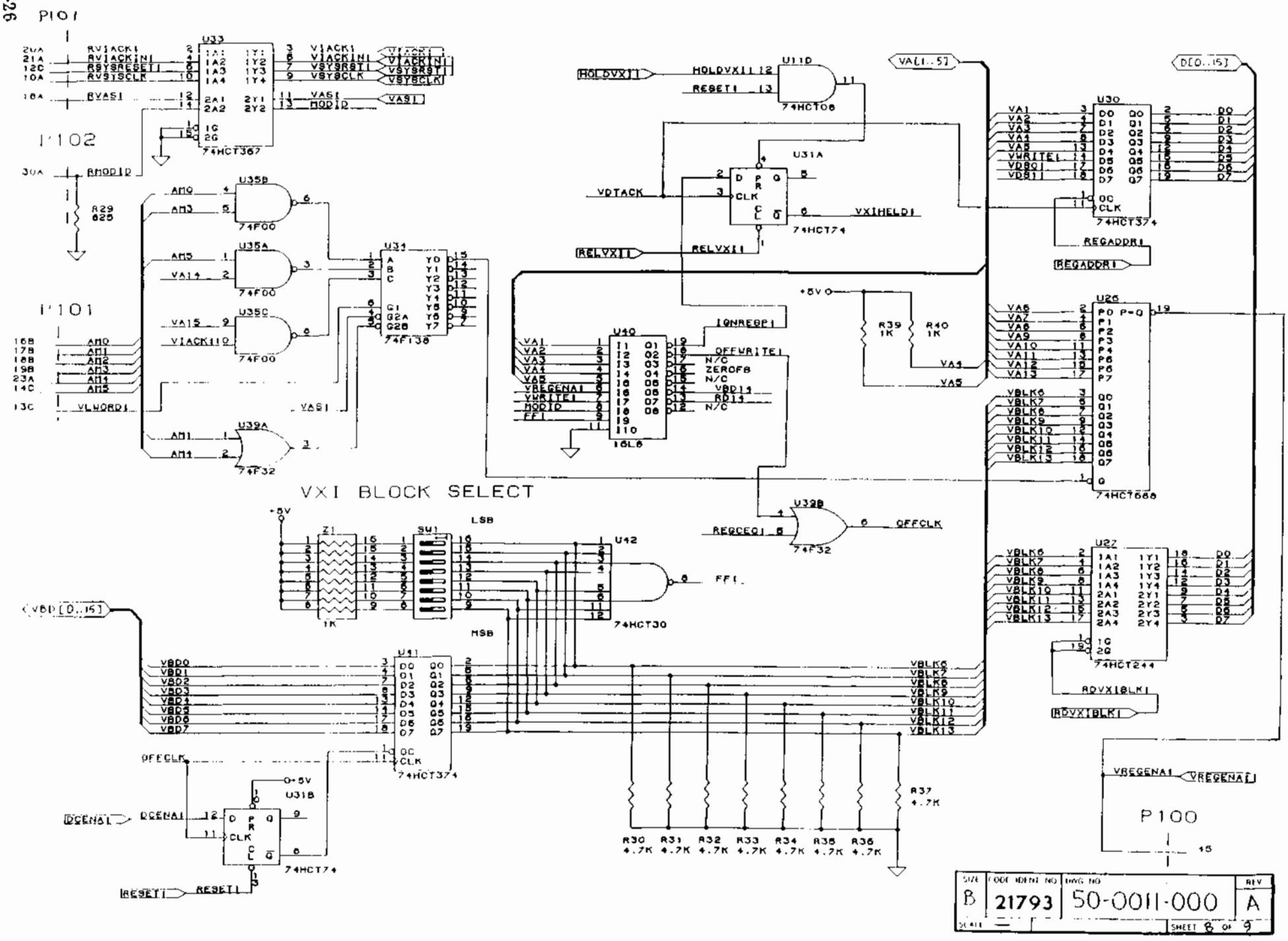

+5V
R16
4.7K
R15
4.7K
R14
4.7K
R13
4.7K
R12
4.7K
R11
4.7K
R10
4.7K
R9
4.7K
U14
CTIRQI
IRQ2I
LBIRQI
IRQ1I
TA1Q1RQI
VXIHELDI
NMII
VXIHELDI
A0
A1
A2
GS
EI
EO
74HC148
R19
1K

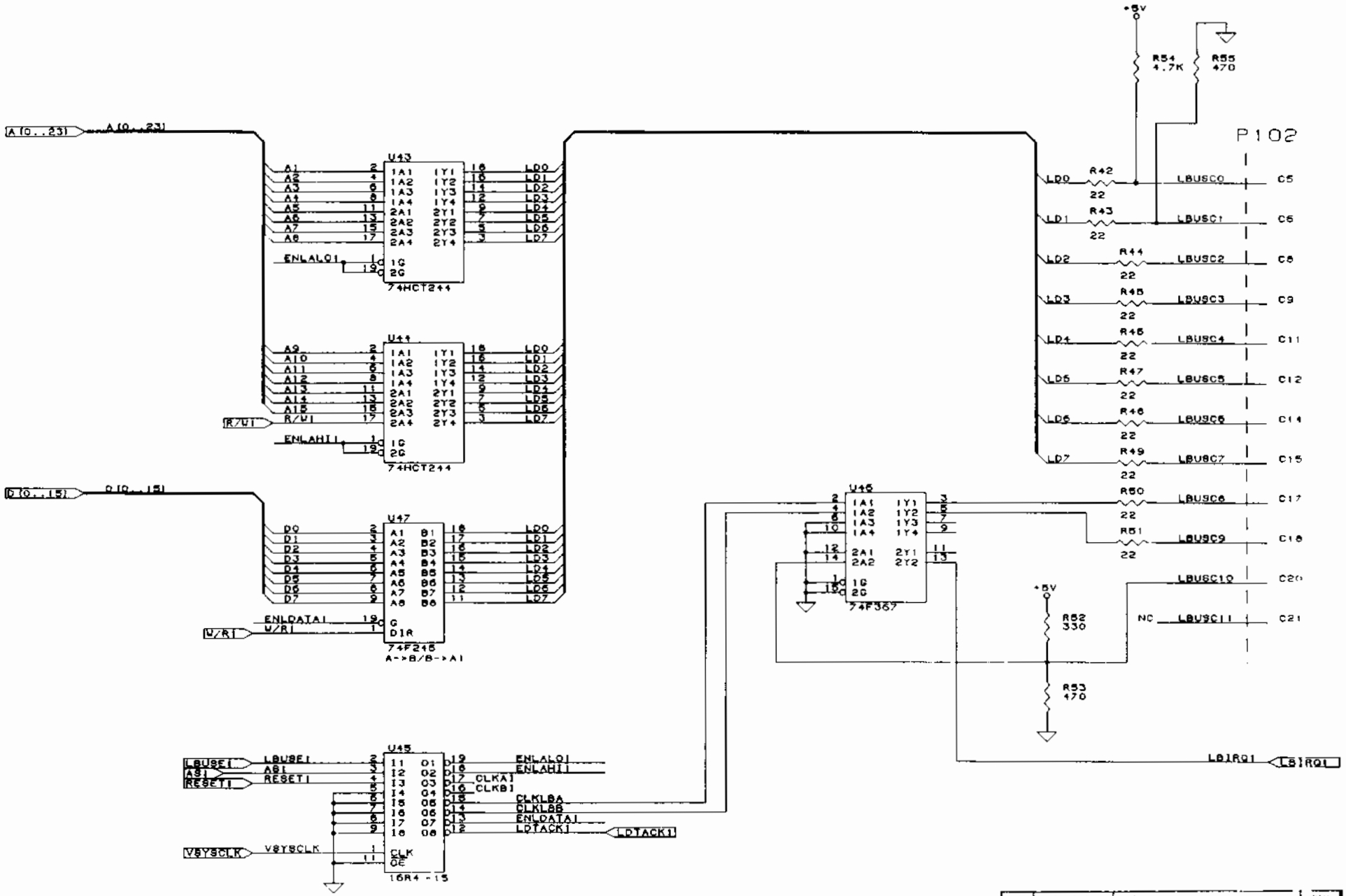




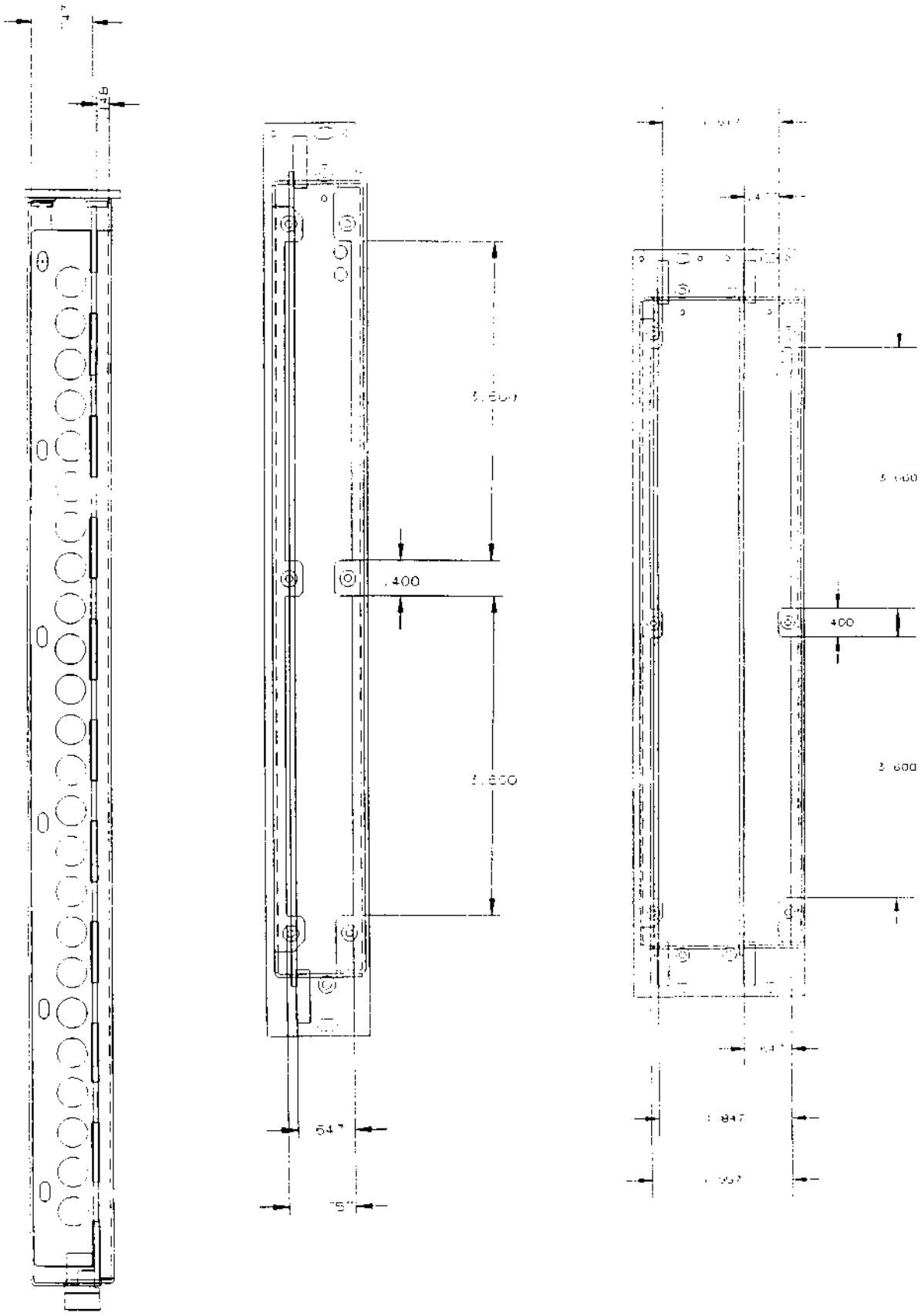
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0011-000	A
SCALE —		SHEET 6 OF 9	

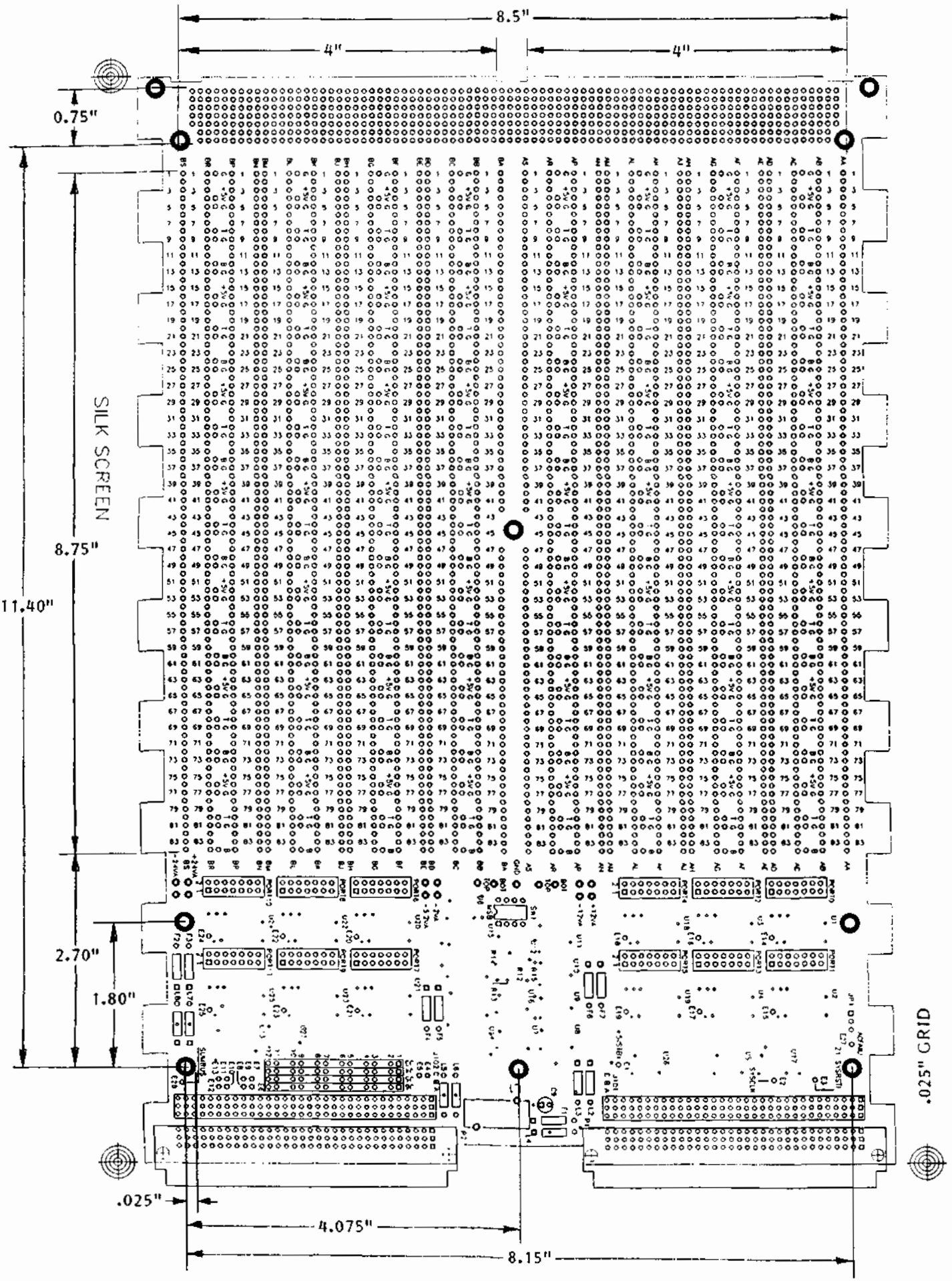

P101
U37
74F652
U38
74F652
U32
74HCT367
U36
22V10FN
U28
IDT71308AJ
ODD REGISTERS
U29
IDT71308AJ
EVEN REGISTERS
+5V
R41
1K
R38
1K
D[0..15]
A[0..23]
VXIBUSY[]
LR/WI
REGENAI
REGBUSYI
VBD[0..15]
VA[1..5]
ACKEDI
RESETI
VXI REGISTER MAI
3FH DEVICE DEPENDANT REGISTERS
1FH
18H VXI RESERVED REGISTERS
14H A32 POINTER
10H A24 POINTER
0EH DATA LOW
0CH DATA HIGH
0AH RESPONSE/DATA EXTENDED
08H PROTOCOL/SIGNAL REGISTER
06H OFFSET REGISTER
04H STATUS/CONTROL REGISTER
02H DEVICE TYPE
00H ID/LOGICAL ADDRESS REGISTER
SIZE B
CODE IDENT NO 21793
DWG NO 50-0011-000
REV A
SCALE —
SHEET 7 OF 9

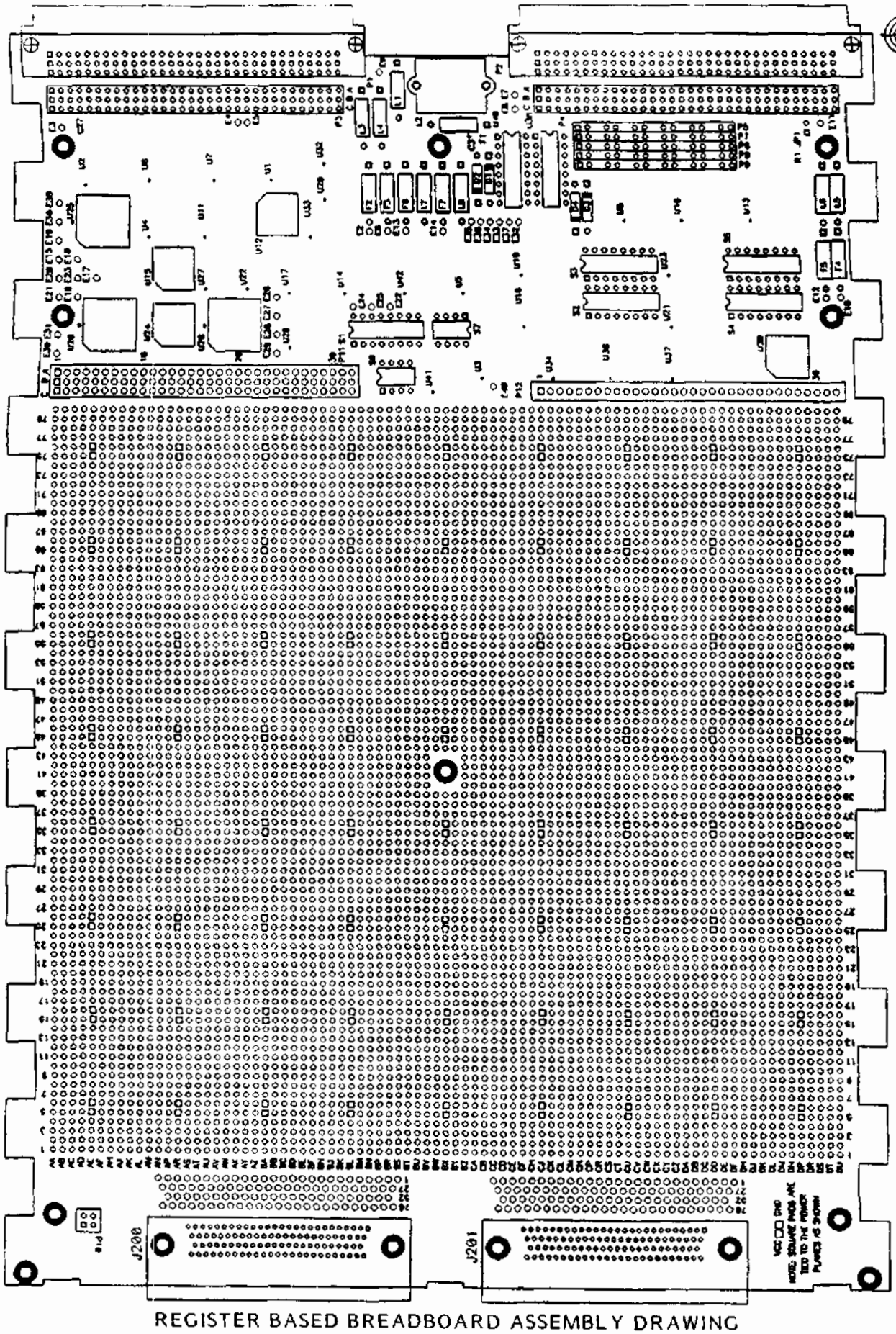

P101
P102
P101
U33
74HCT367
U35B
74F00
U35A
74F00
U35C
74F00
U34
74F138
U39A
74F32
R29
625
U11D
74HCT08
U31A
74HCT74
U30
74HCT374
U26
74HCT688
U27
74HCT244
U40
16L8
U39B
74F32
VXI BLOCK SELECT
Z1
1K
SW1
LSB
MSB
U42
74HCT30
U41
74HCT374
U31B
74HCT74
R39
1K
R40
1K
R37
4.7K
R30
4.7K
R31
4.7K
R32
4.7K
R33
4.7K
R34
4.7K
R35
4.7K
R36
4.7K
P100
VA[1..5]
D[0..15]
VBD[0..15]
RESET1
RDVXIBLK1
REGADDR1
VREGENA1
SIZE
B
CODE IDENT NO
21793
DWG NO
50-0011-000
REV
A
SHEET 8 OF 9



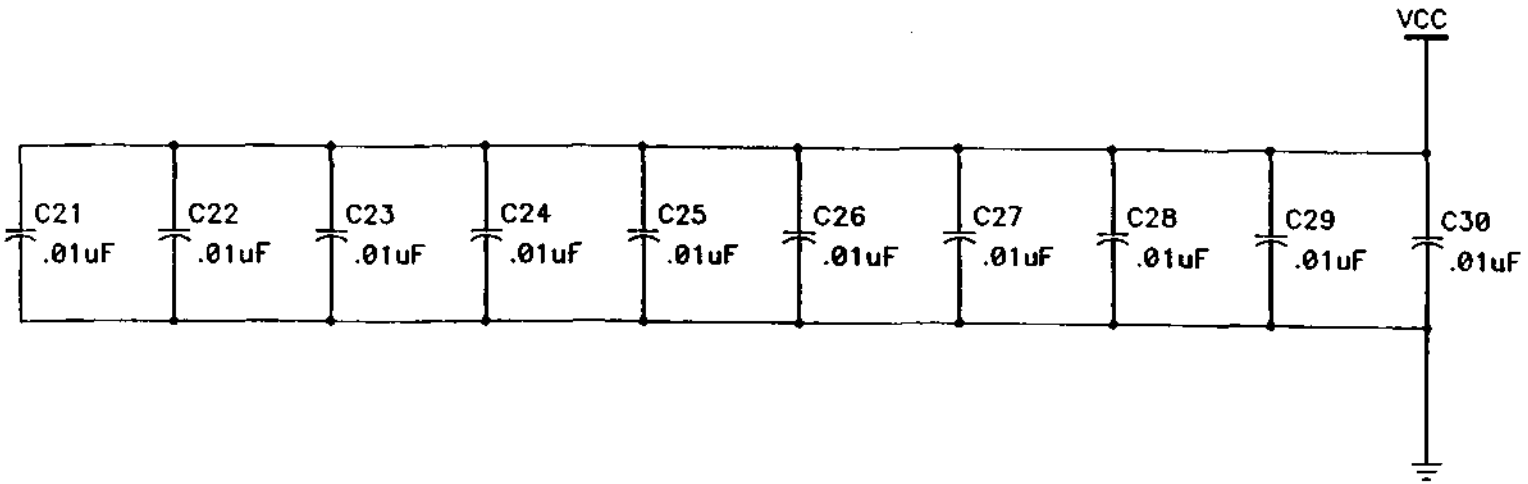
SIZE	CODE IDENT NO	DWG NO	REV
B	21793	50-0011-000	A
SCALE —		SHEET 9 OF 9	


.400
.400


8.5"
4"
4"
0.75"
SILK SCREEN
8.75"
11.40"
2.70"
1.80"
.025" GRID
.025"
4.075"
8.15"



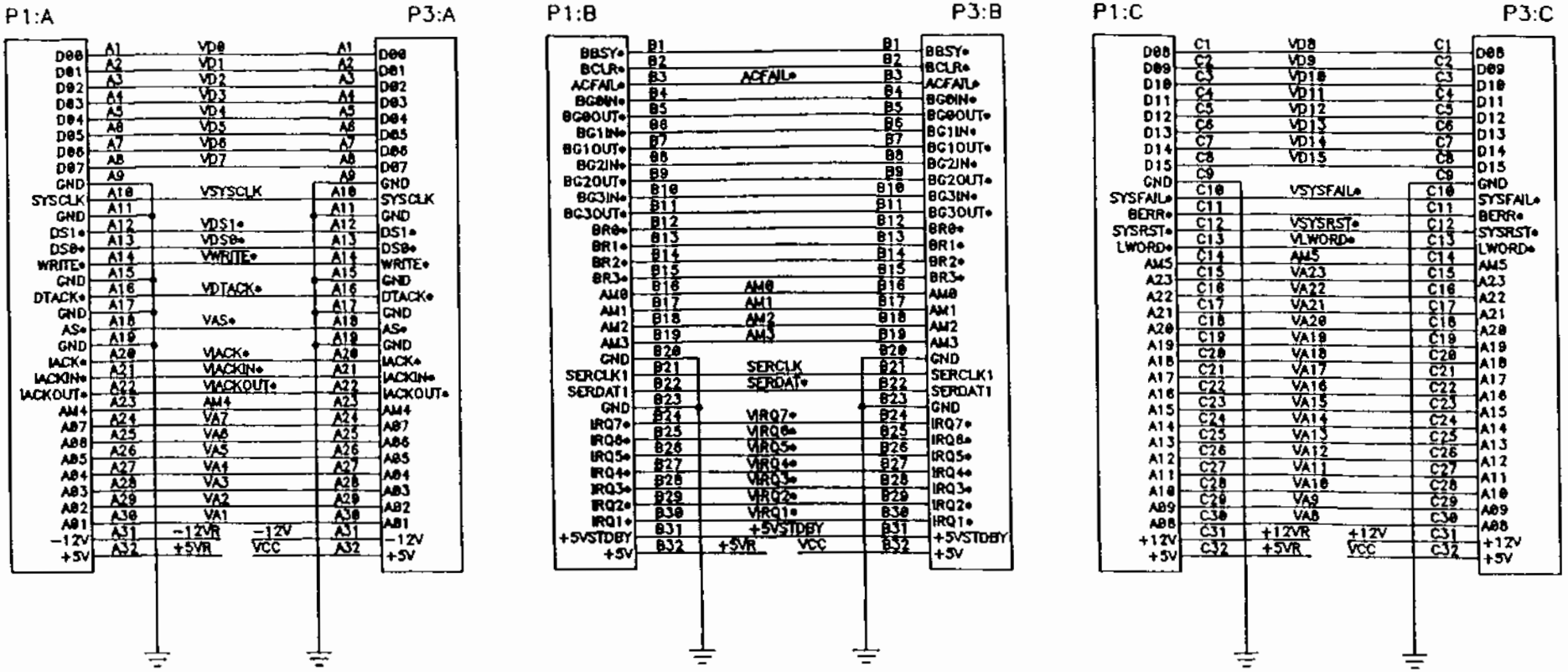
REGISTER BASED BREADBOARD ASSEMBLY DRAWING



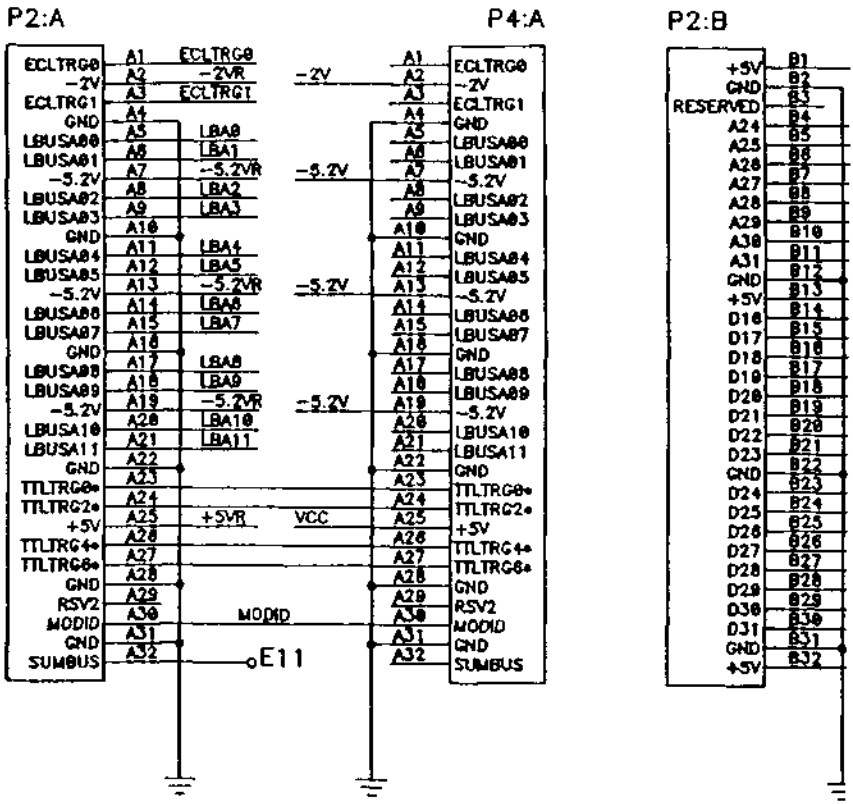
6. INSTALL A WIRE JUMPER FROM E38 TO E39
5. REFER TO 33-0023-000/001 FOR PCB
4. REFER TO 52-0027-000/001 FOR ASSEMBLY DRAWINGS
3. ALL CAPS LESS THAN 1.0uF ARE 50V 20% MONOLYTHIC
2. ALL CAPS GREATER THAN 1.0uF ARE 16V 20% TANTALUM
1. ALL RESISTORS ARE .06W 5%

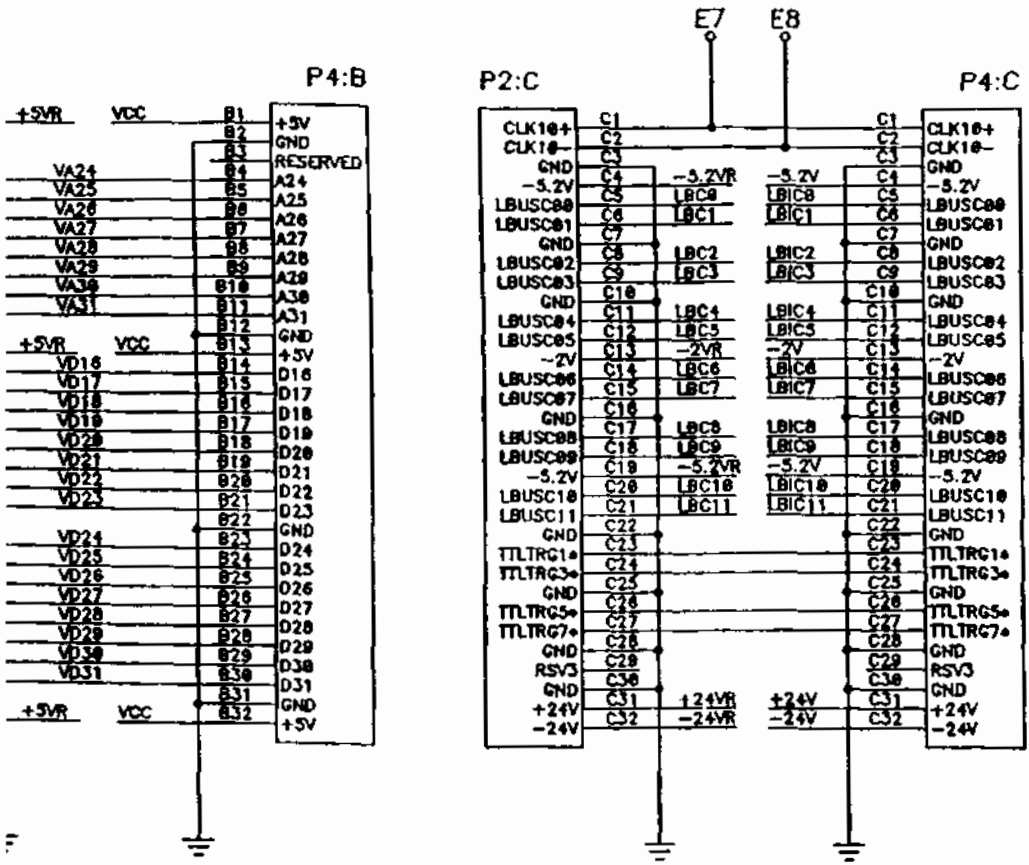
NOTES: UNLESS OTHER WISE SPECIFIED

TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S01	SHEET: 1 OF 19	



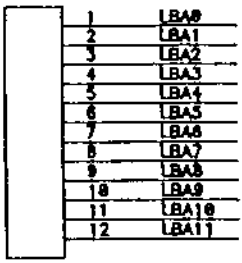
TITLE	SCHEMATIC, REGISTER BASED BREADBOARD		
SIZE C	CAGE CODE	DWG NO. 50-0026-000	REV. A
SCALE NONE	FILE 0026-000.S02	SHEET 2 OF 19	


P2:A
P4:A
P2:B
ECLTRG0
-2V
ECLTRG1
GND
LBUSA00
LBUSA01
-5.2V
LBUSA02
LBUSA03
GND
LBUSA04
LBUSA05
-5.2V
LBUSA06
LBUSA07
GND
LBUSA08
LBUSA09
-5.2V
LBUSA10
LBUSA11
GND
TTLTRG0*
TTLTRG2*
+5V
TTLTRG4*
TTLTRG6*
GND
RSV2
MODID
GND
SUMBUS
-2VR
LBA0
LBA1
-5.2VR
LBA2
LBA3
LBA4
LBA5
-5.2VR
LBA6
LBA7
LBA8
LBA9
-5.2VR
LBA10
LBA11
+5VR
VCC
MODID
E11
-2V
-5.2V
-5.2V
-5.2V
+5V
GND
RESERVED
A24
A25
A26
A27
A28
A29
A30
A31
GND
+5V
D16
D17
D18
D19
D20
D21
D22
D23
GND
D24
D25
D26
D27
D28
D29
D30
D31
GND
+5V

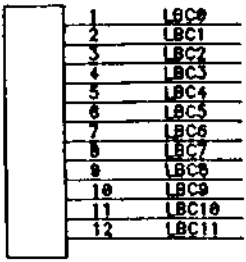


TITLE	SCHEMATIC, REGISTER BASED BREADBOARD		
SIZE C	CAGE CODE	DWG NO. 50-0026-000	REV. A
SCALE NONE	FILE 0026-000.S02	SHEET 3 OF 19	

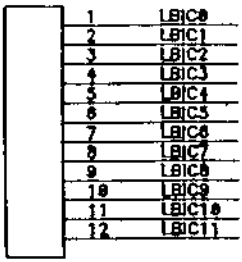
P5


1 LBA0
2 LBA1
3 LBA2
4 LBA3
5 LBA4
6 LBA5
7 LBA6
8 LBA7
9 LBA8
10 LBA9
11 LBA10
12 LBA11

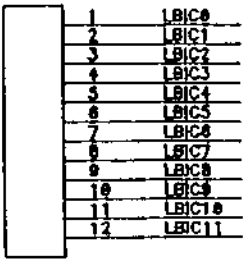
P6

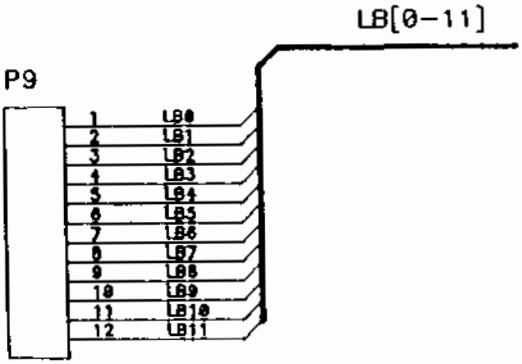

1 LBC0
2 LBC1
3 LBC2
4 LBC3
5 LBC4
6 LBC5
7 LBC6
8 LBC7
9 LBC8
10 LBC9
11 LBC10
12 LBC11

P7

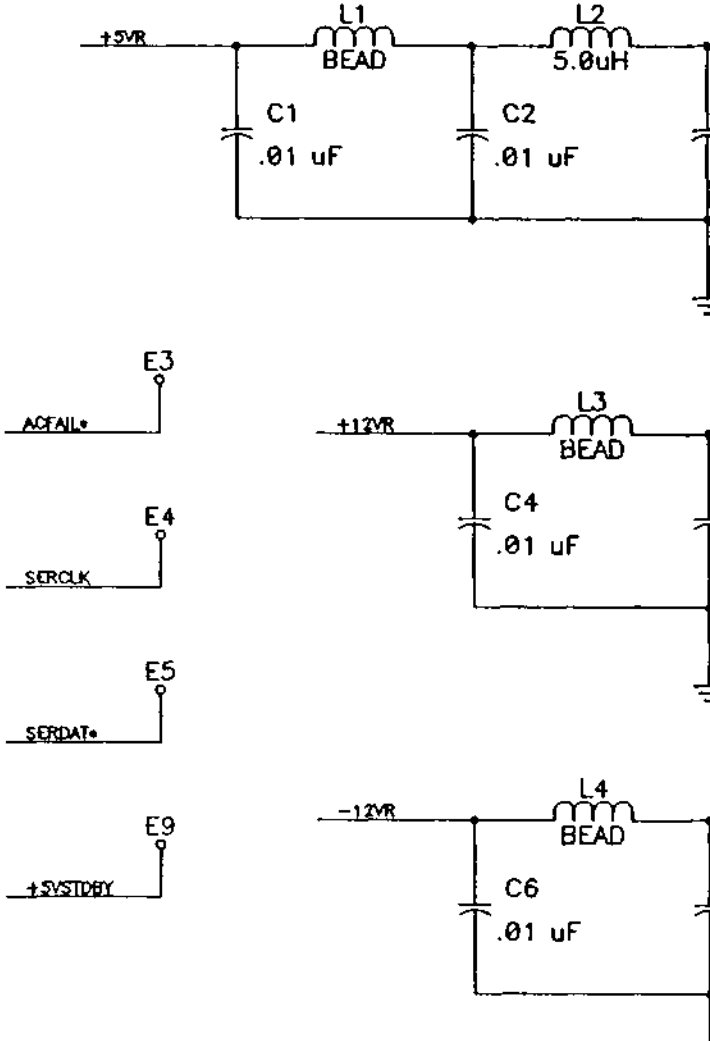

1 LBIC0
2 LBIC1
3 LBIC2
4 LBIC3
5 LBIC4
6 LBIC5
7 LBIC6
8 LBIC7
9 LBIC8
10 LBIC9
11 LBIC10
12 LBIC11

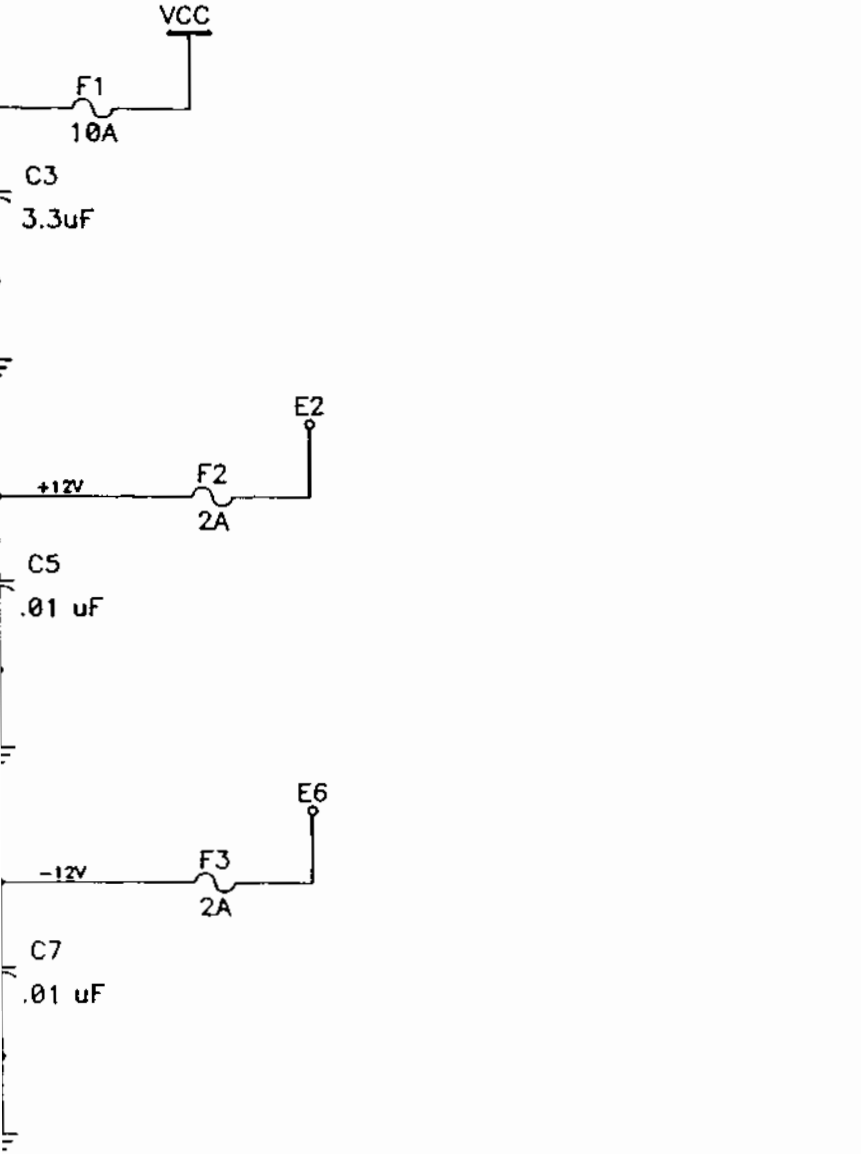
P8


1 LBIC0
2 LBIC1
3 LBIC2
4 LBIC3
5 LBIC4
6 LBIC5
7 LBIC6
8 LBIC7
9 LBIC8
10 LBIC9
11 LBIC10
12 LBIC11

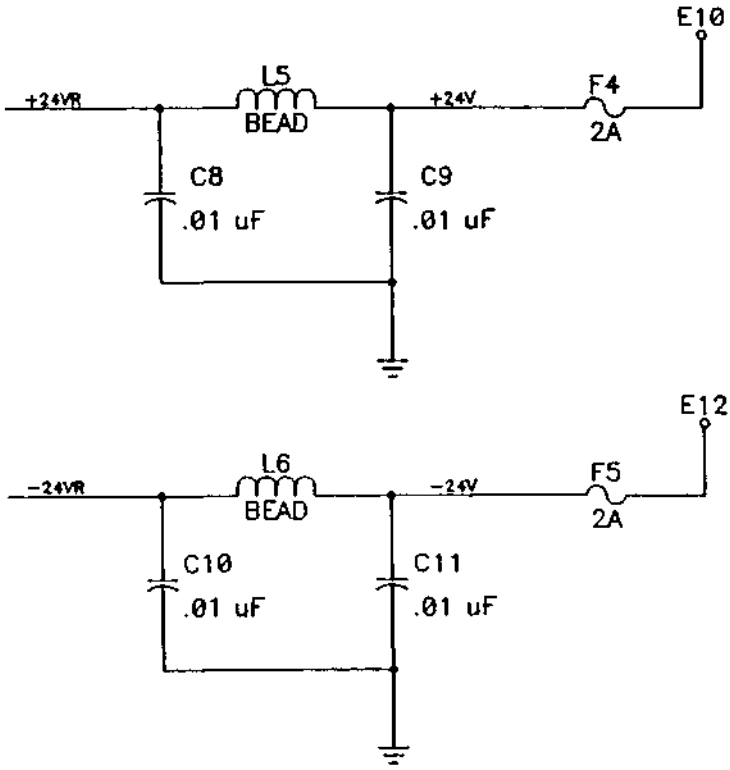


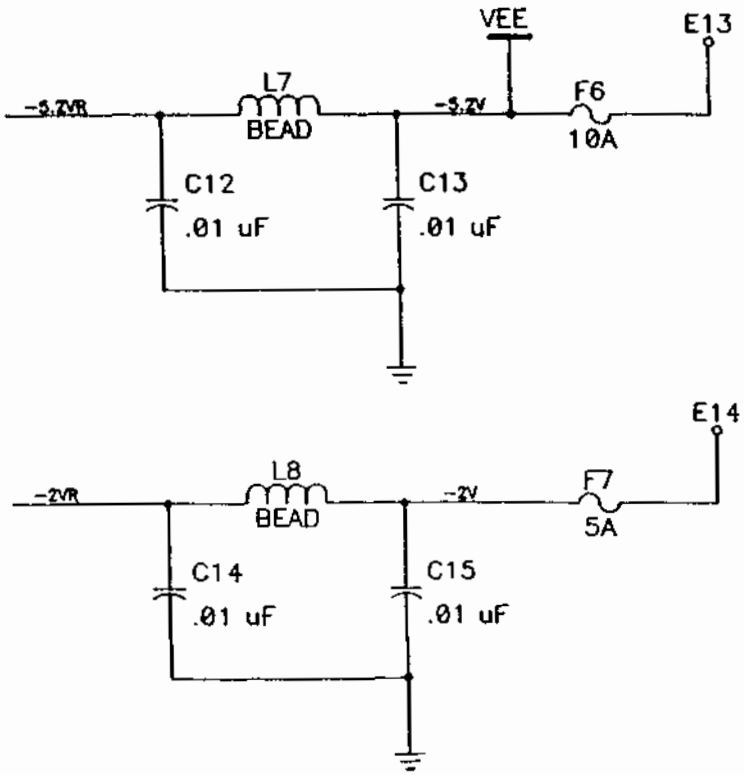
TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO.: 50-0026-000	REV.: A
SCALE: NONE	FILE: 0026-000.S02	SHEET: 4 OF 19	


+5VR
L1
BEAD
L2
5.0uH
C1
.01 uF
C2
.01 uF
E3
ACFAIL*
E4
SERCLK
E5
SERDAT*
E9
+5VSTDBY
+12VR
L3
BEAD
C4
.01 uF
-12VR
L4
BEAD
C6
.01 uF

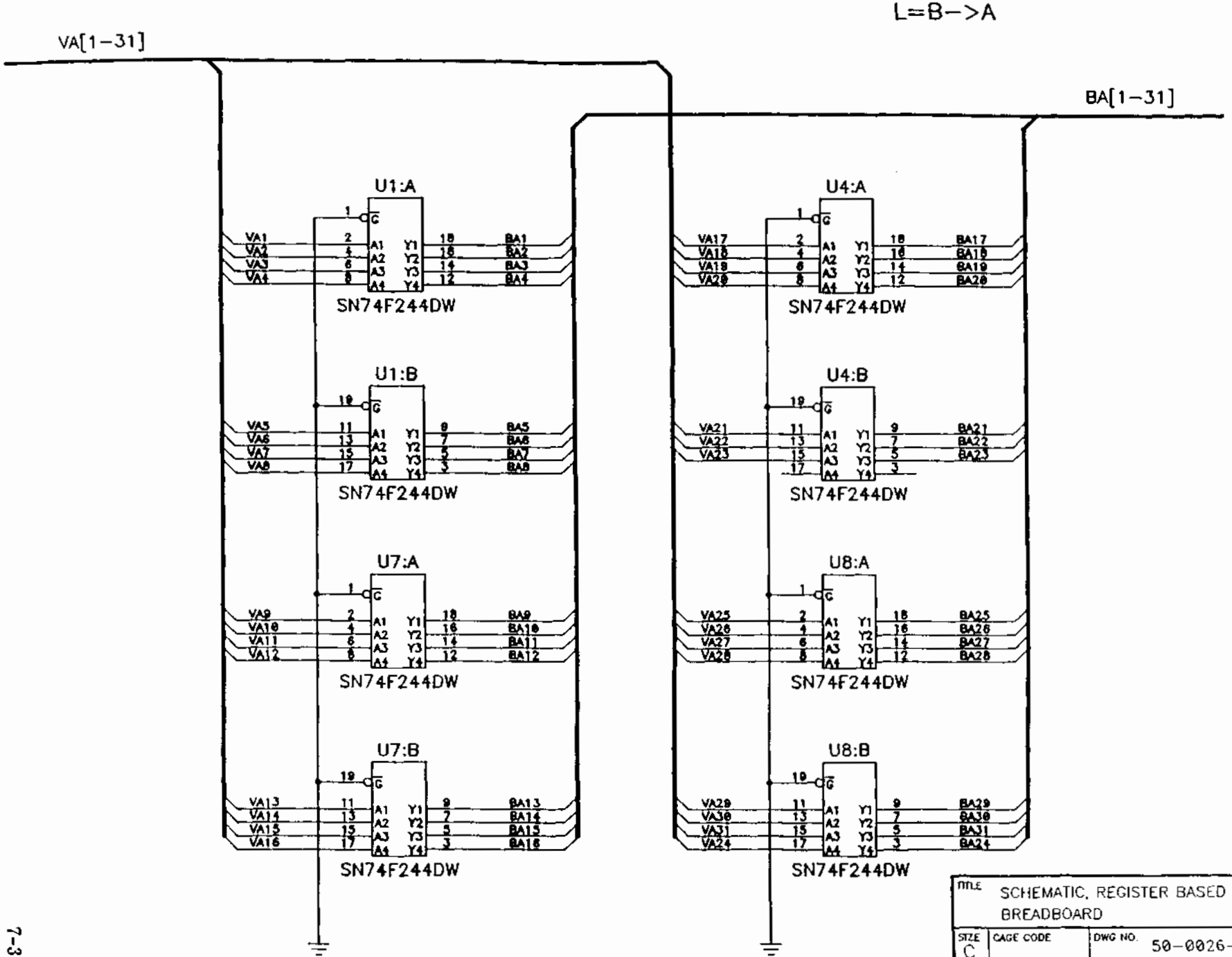


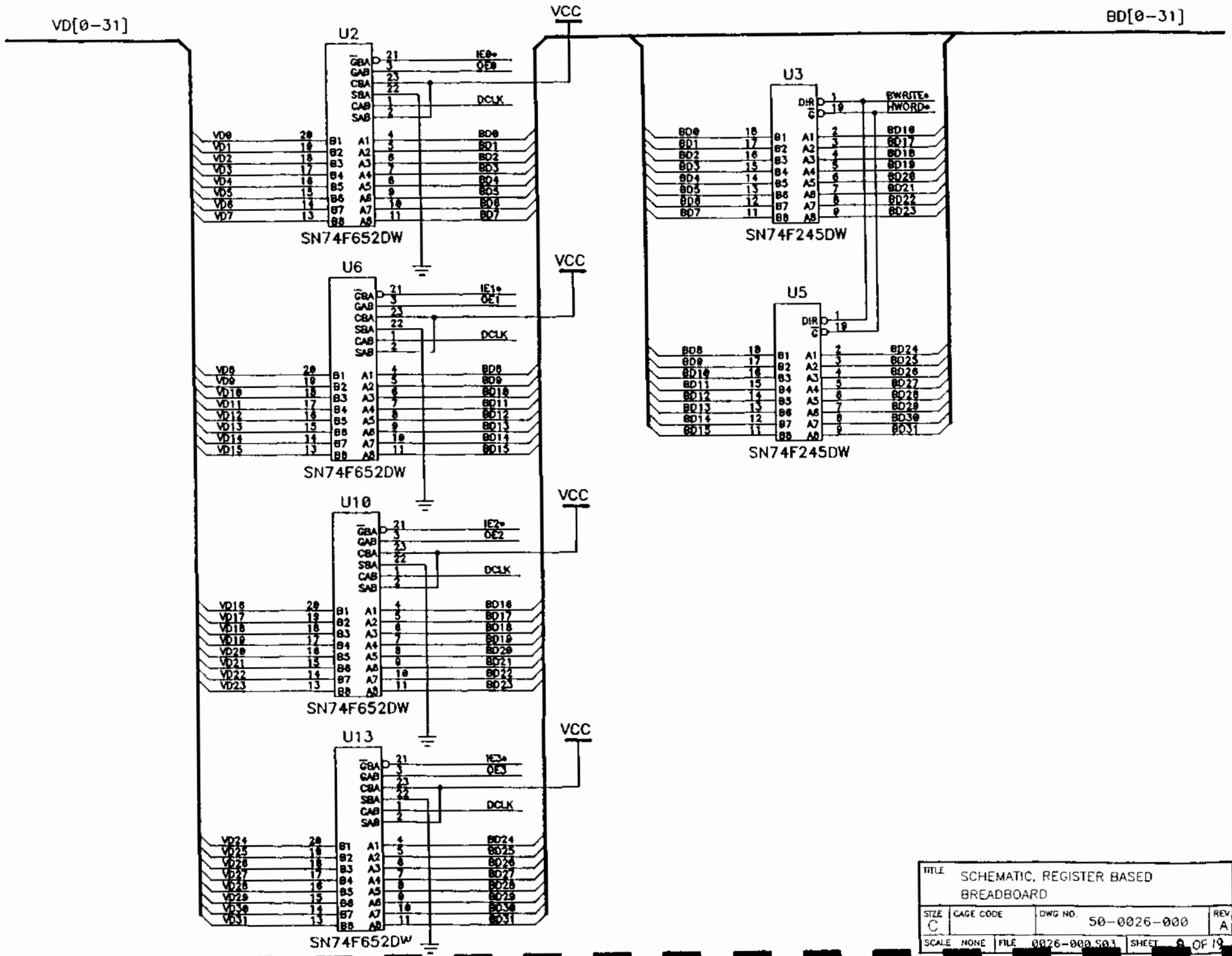
TITLE	SCHEMATIC, REGISTER BASED BREADBOARD		
SIZE C	CAGE CODE	DWG NO. 50-0026-000	REV. A
SCALE NONE	FILE 0026-000.S02	SHEET 5 OF 19	


E10
+24VR
L5
BEAD
+24V
F4
2A
C8
.01 uF
C9
.01 uF
E12
-24VR
L6
BEAD
-24V
F5
2A
C10
.01 uF
C11
.01 uF

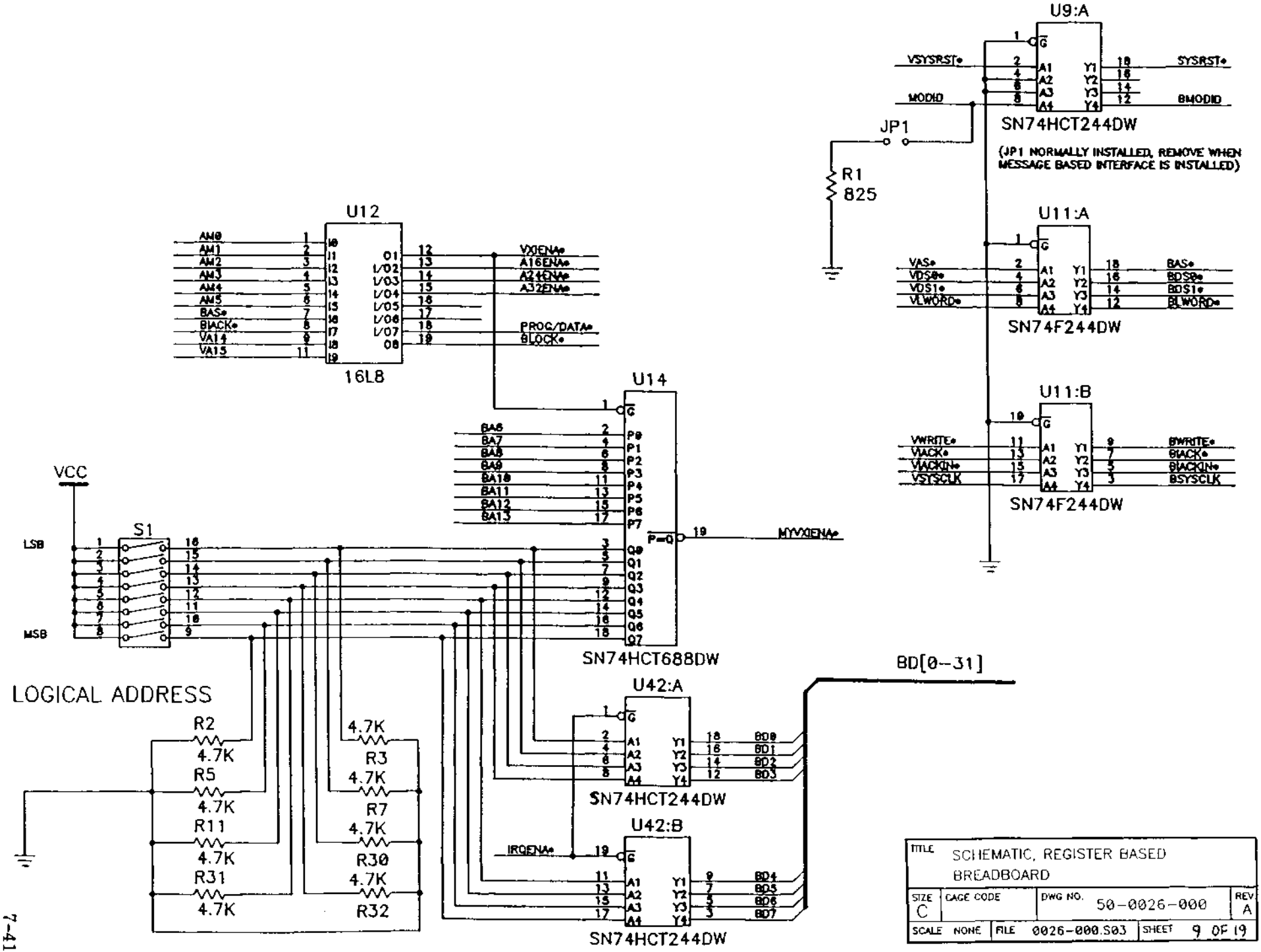


TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO.: 50-0026-000	REV.: A
SCALE: NONE	FILE: 0026-000.S02	SHEET: 6 OF 19	

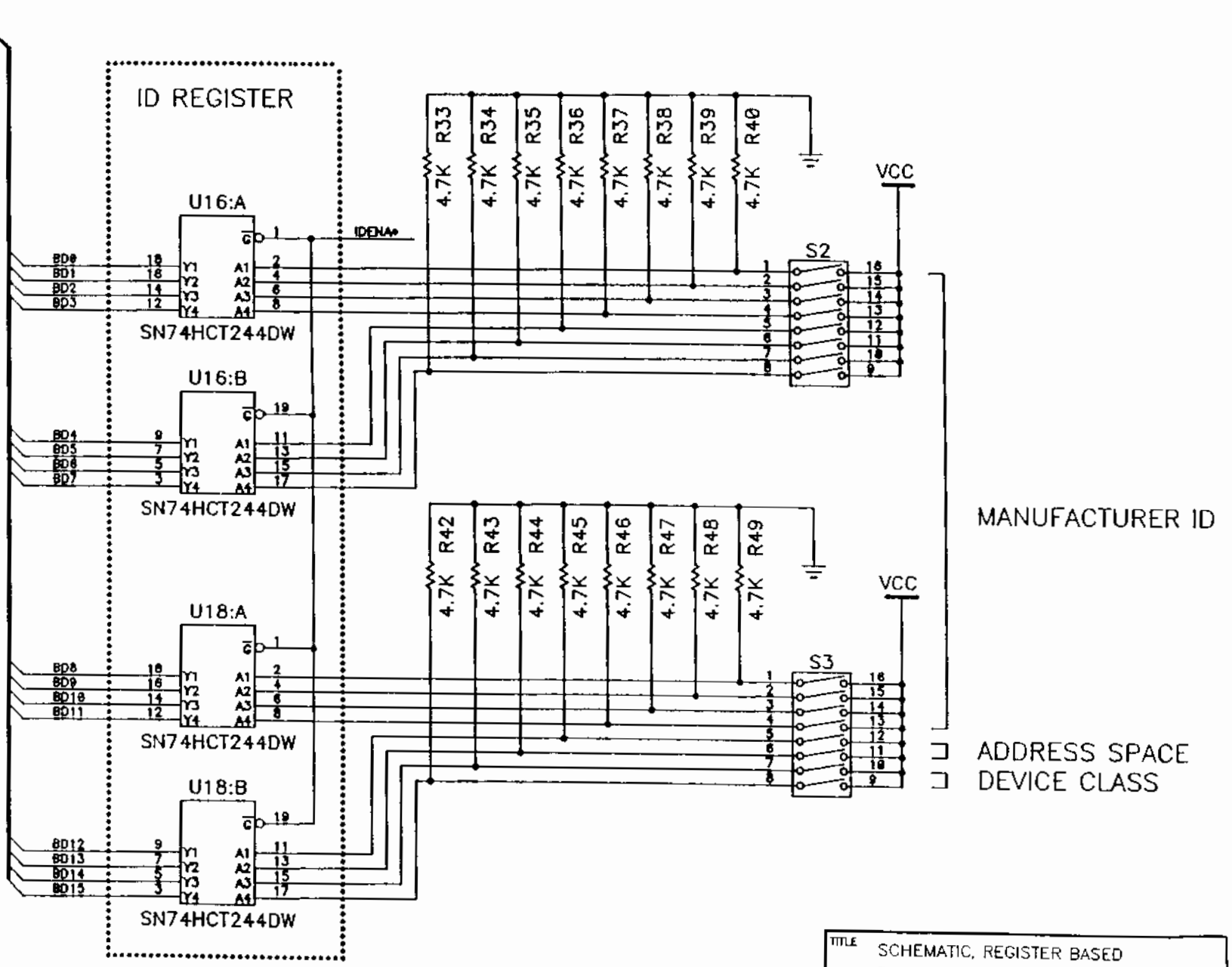

L=B->A
VA[1-31]
BA[1-31]
U1:A
U1:B
U7:A
U7:B
U4:A
U4:B
U8:A
U8:B
SN74F244DW
SCHEMATIC, REGISTER BASED BREADBOARD
SIZE C
CAGE CODE
DWG NO. 50-0026-000
REV A
SCALE NONE
FILE 0026-000 S03
SHEET 7 OF 19



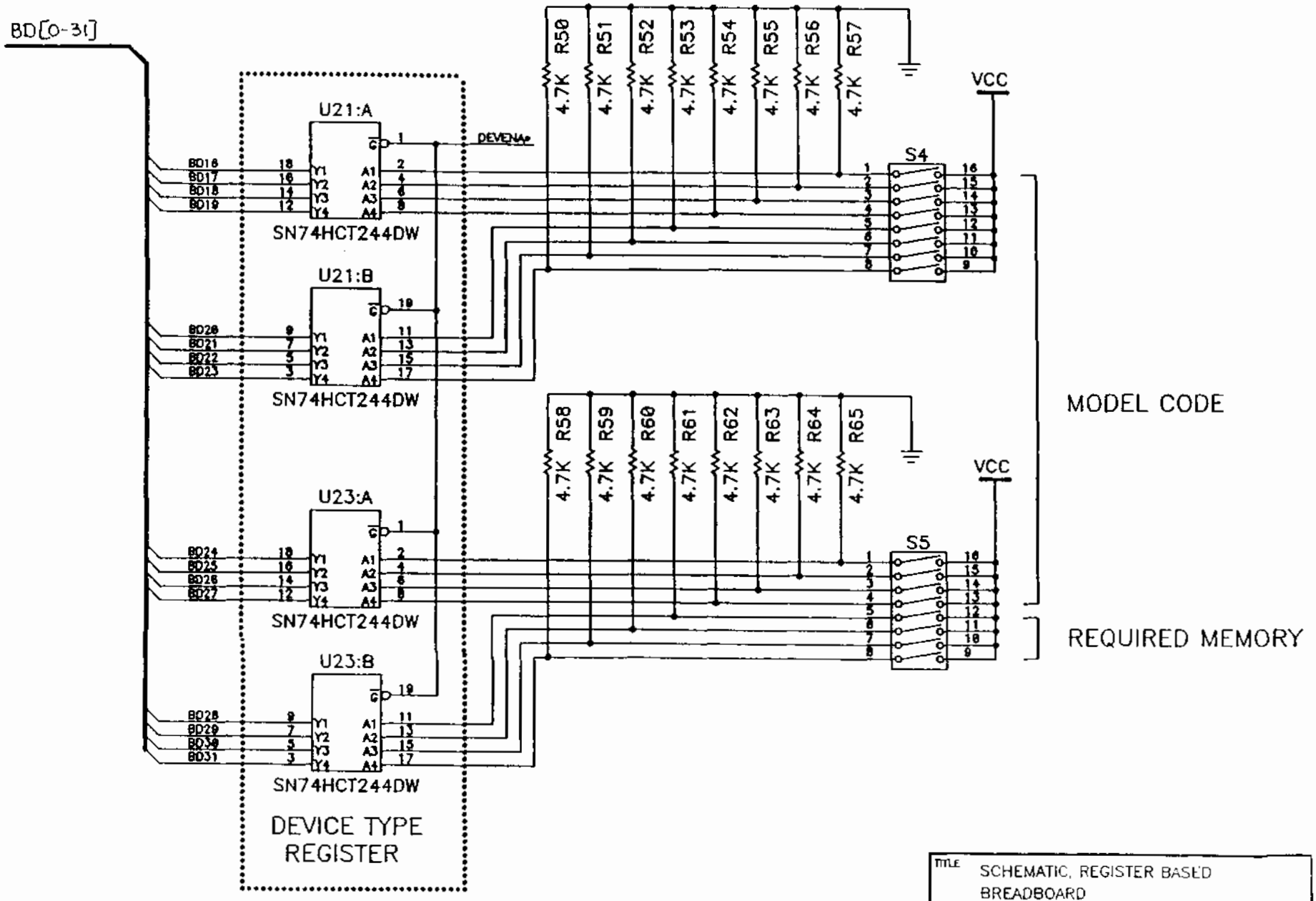
TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO.: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.SCH	SHEET: 8 OF 19	



TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE	DWG NO.: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S03	SHEET: 9 OF 19	

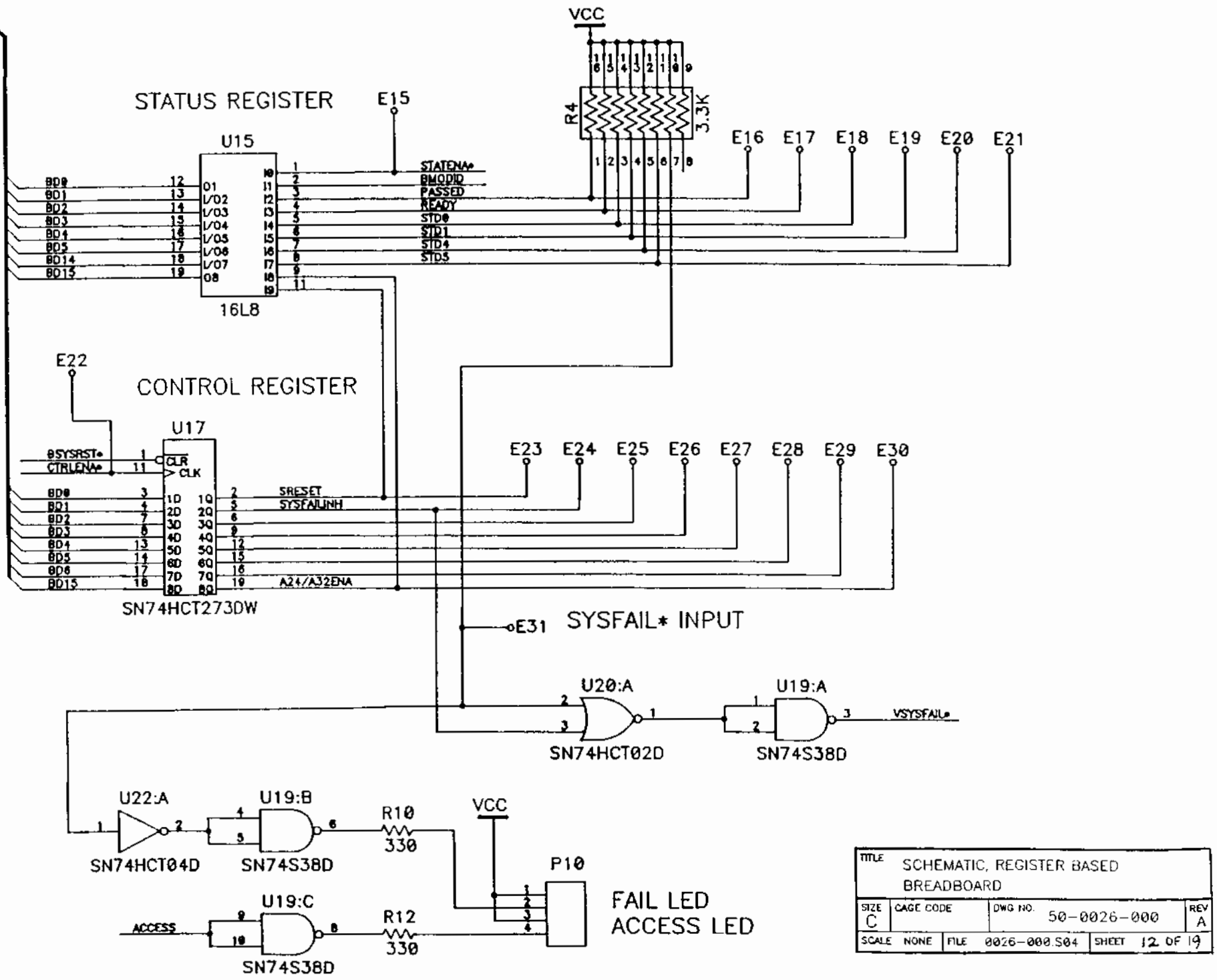


TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO.: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S04	SHEET: 10 OF 19	

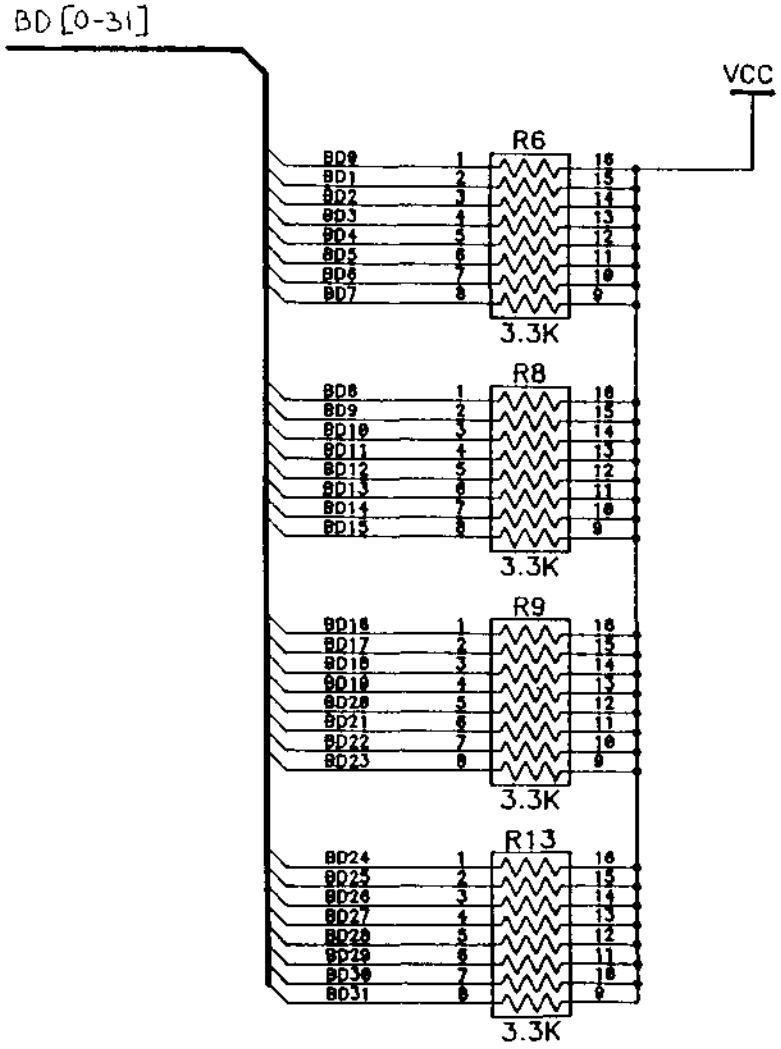


TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S04	SHEET: 11 OF 19	

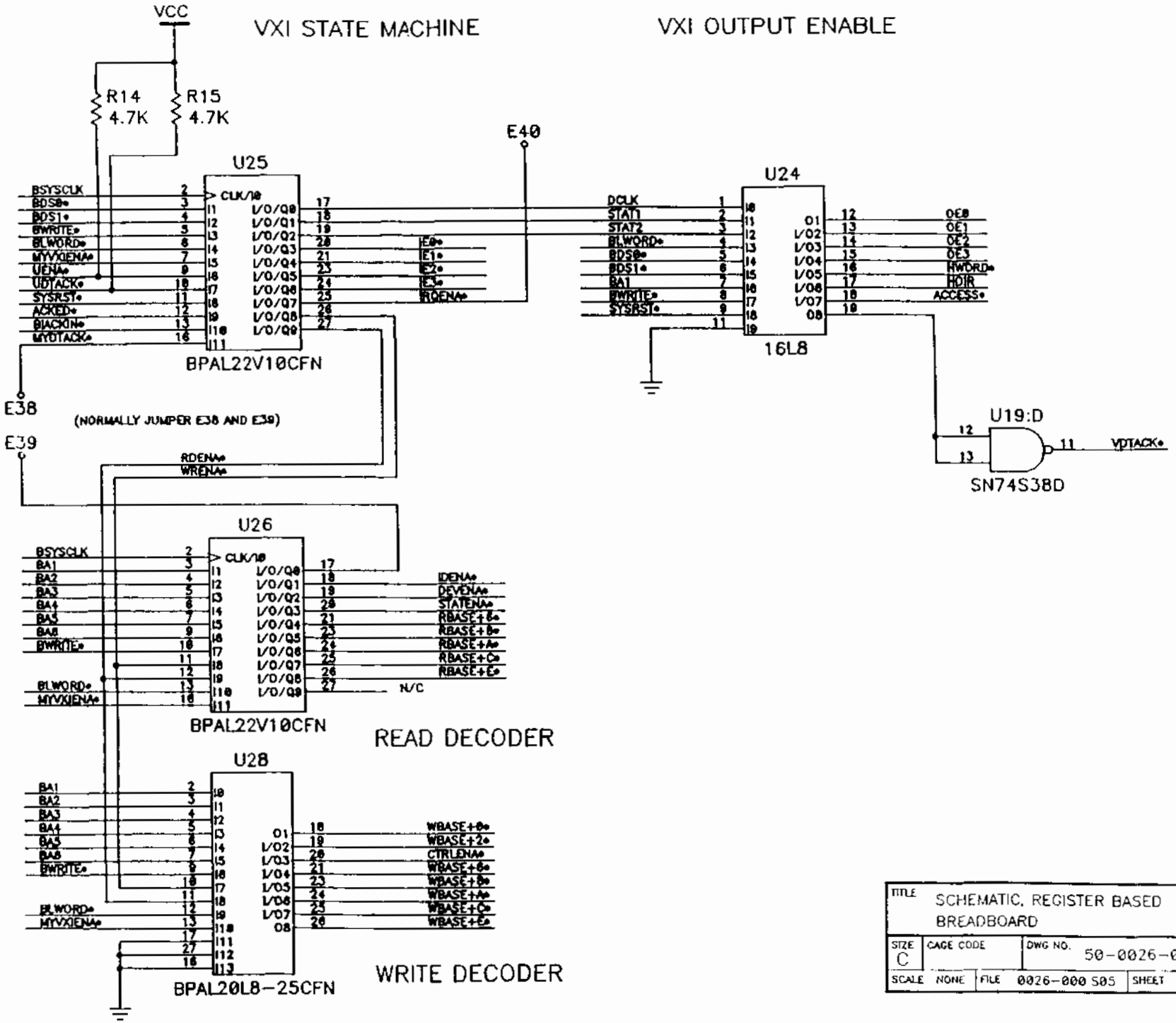
BD[0-31]



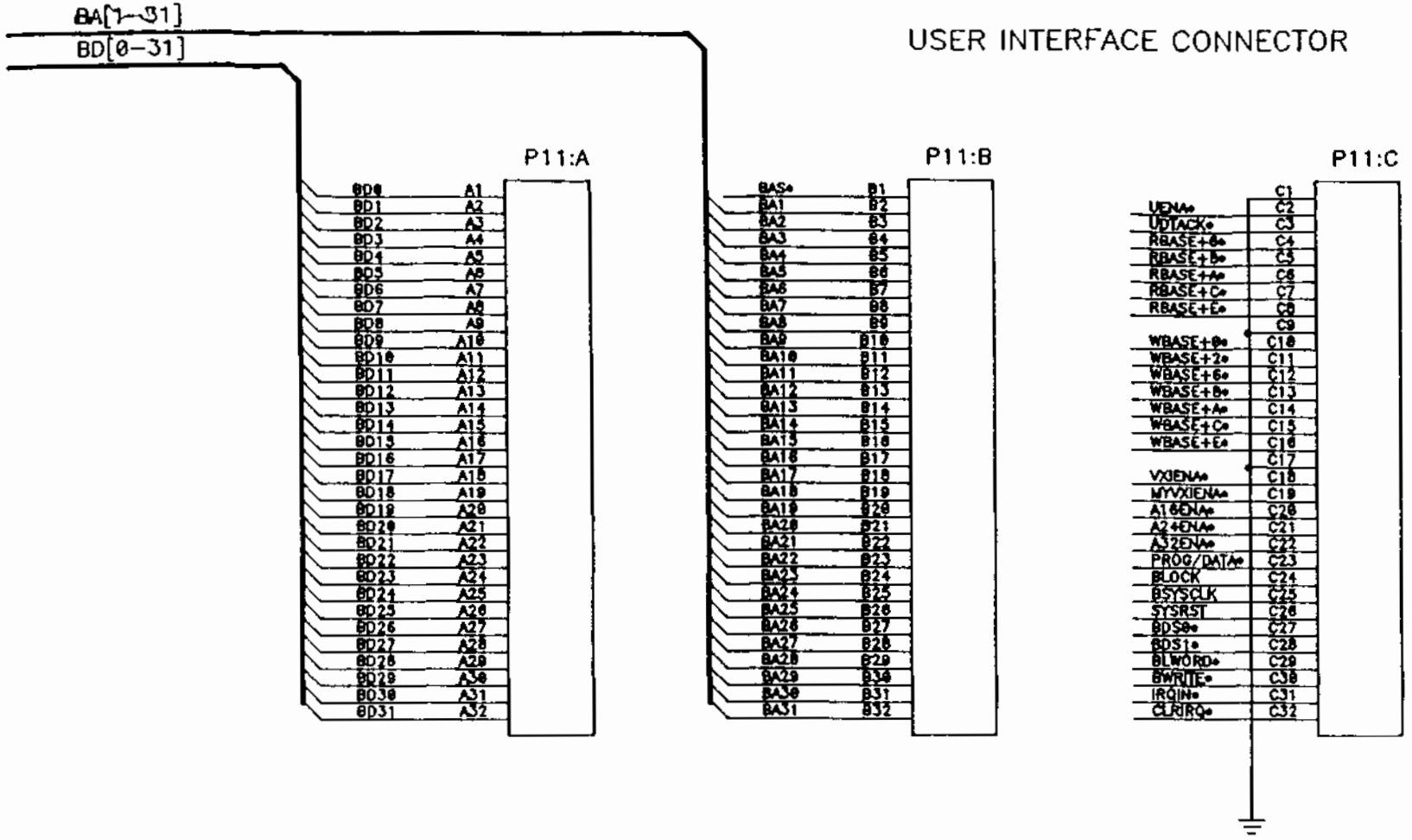
TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE:	DWG NO: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S04	SHEET: 12 OF 19	



TITLE: SCHEMATIC, REGISTER BASED BREADBOARD			
SIZE: C	CAGE CODE	DWG NO.: 50-0026-000	REV: A
SCALE: NONE	FILE: 0026-000.S04	SHEET: 13 OF 19	

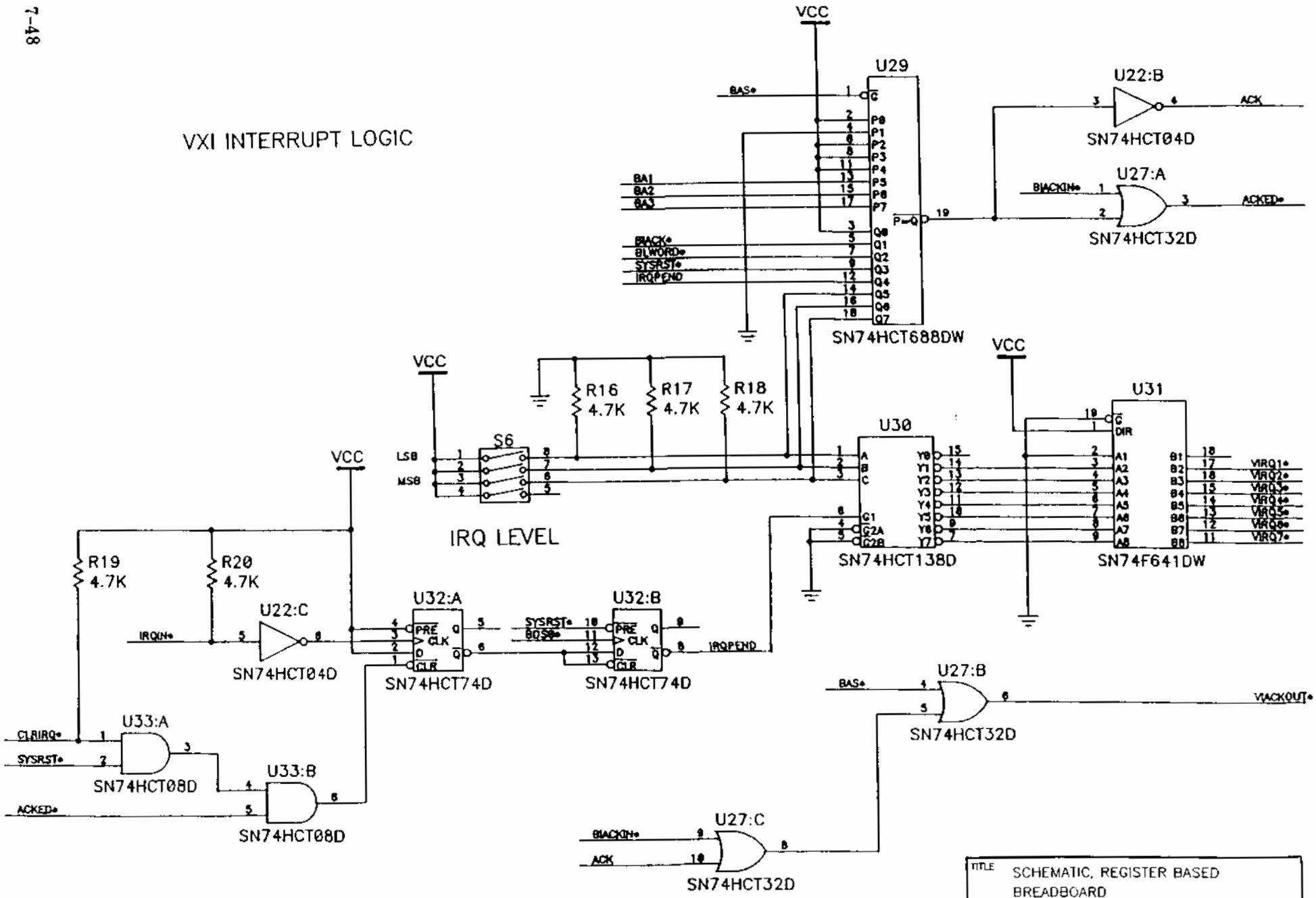


TITLE	SCHEMATIC, REGISTER BASED BREADBOARD		
SIZE C	CAGE CODE	DWG NO. 50-0026-000	REV A
SCALE NONE	FILE 0026-000 S05	SHEET 14 OF 19	

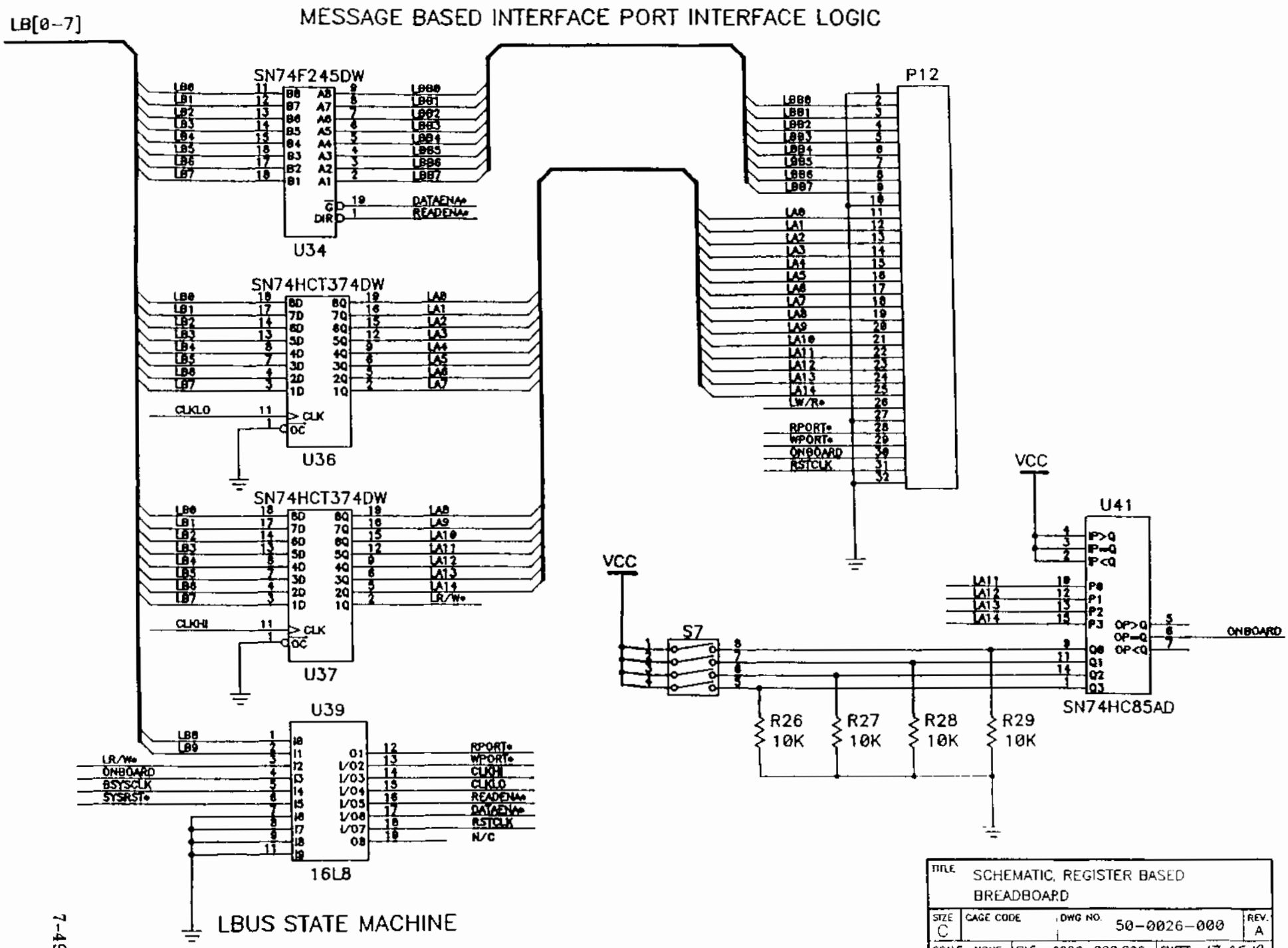


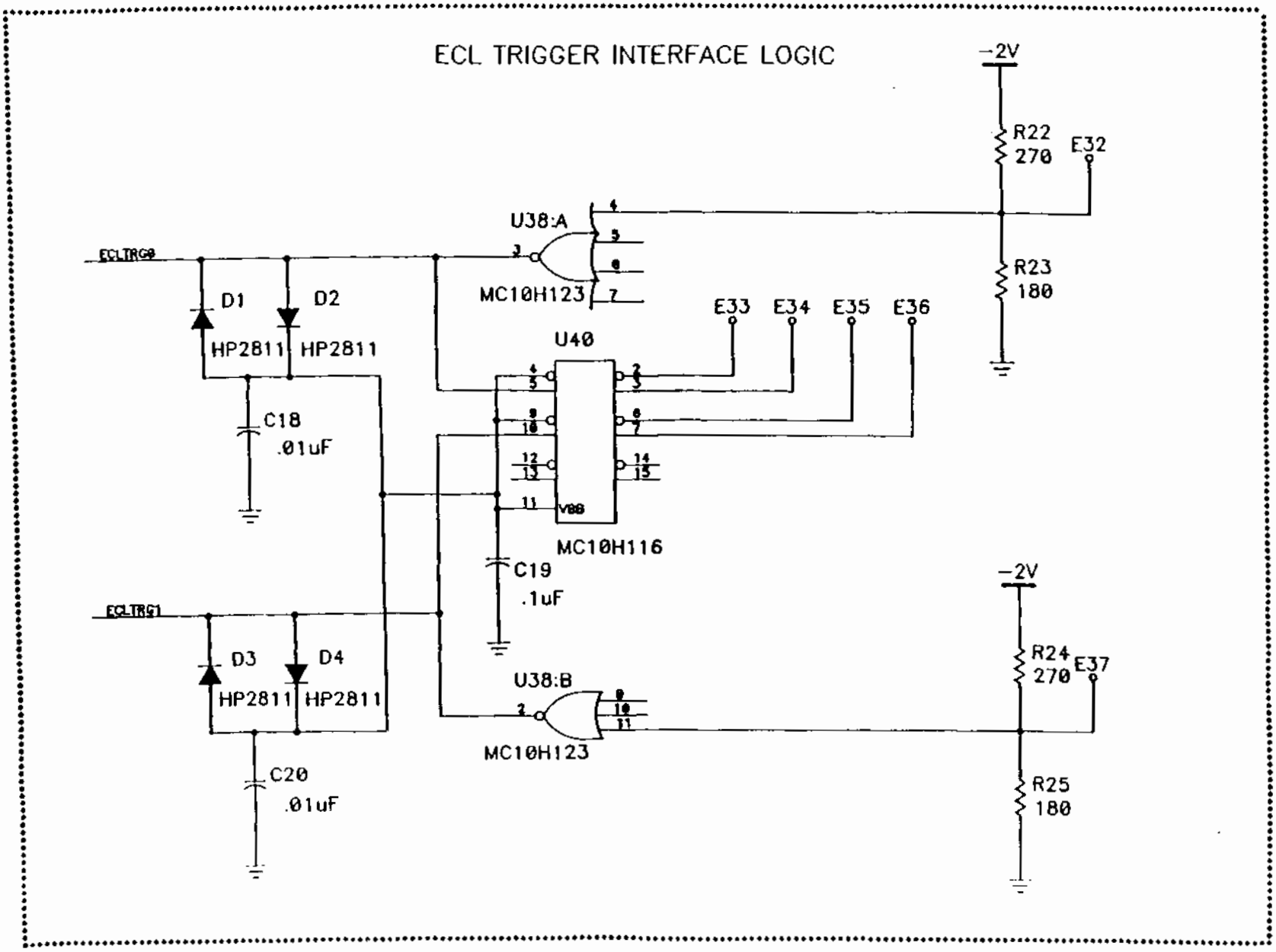
TITLE: SCHEMATIC, REGISTER BASED BREADBOARD				
SIZE: C	CAGE CODE	DWG NO: 50-0026-000		REV: A
SCALE: NONE	FILE: 0026-000.S05		SHEET: 15 OF 19	

VXI INTERRUPT LOGIC

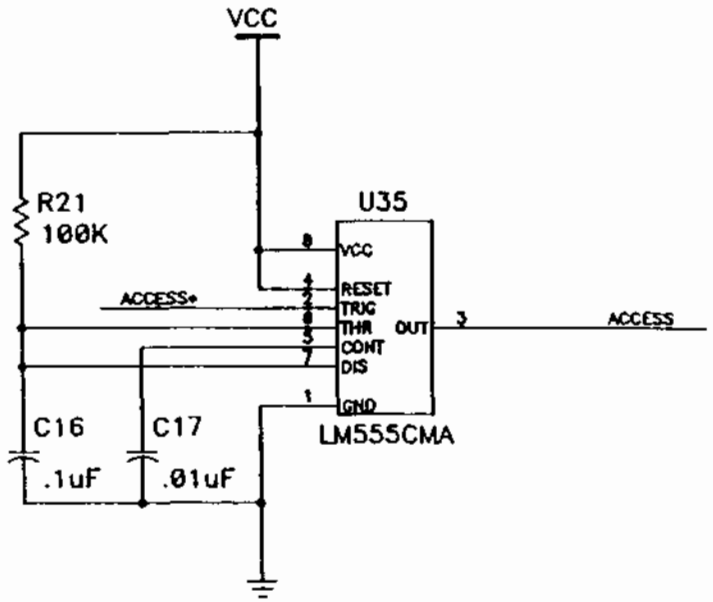


TITLE	SCHEMATIC, REGISTER BASED BREADBOARD		
SIZE C	CAGE CODE	DWG NO. 50-0026-000	REV A
SCALE NONE	FILE 0026-000.SCH	SHEET 16 OF 19	

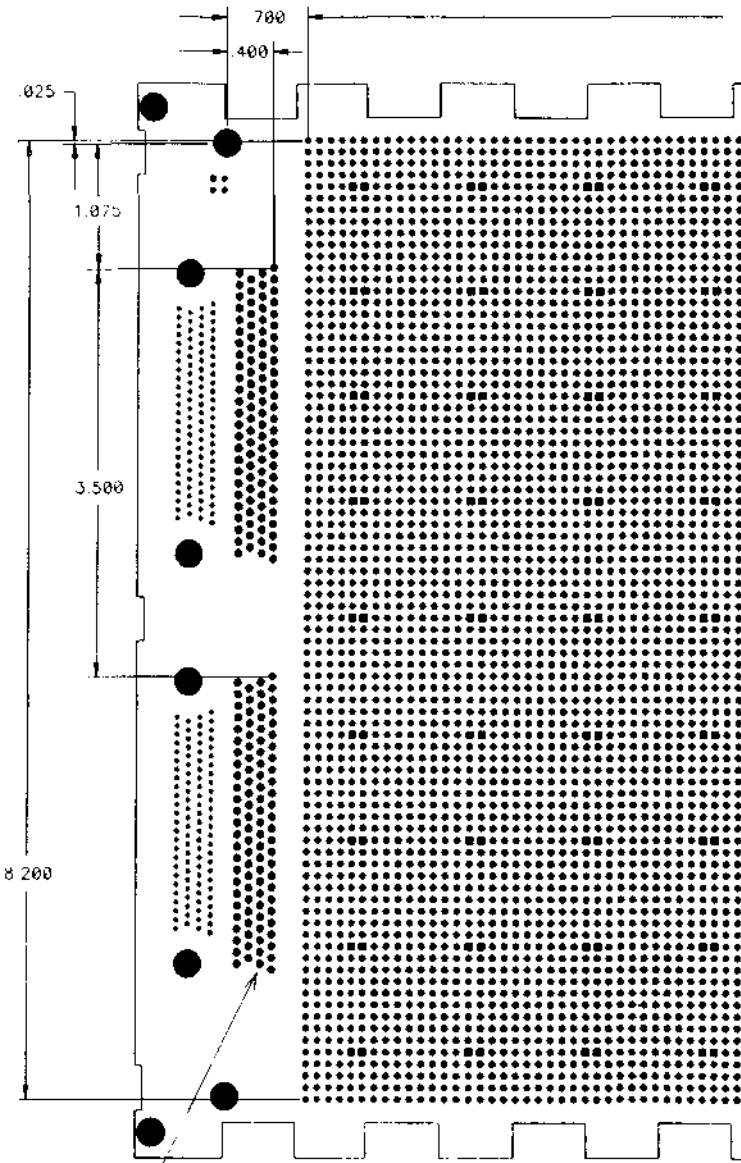

LB[0-7]
MESSAGE BASED INTERFACE PORT INTERFACE LOGIC
SN74F245DW
U34
SN74HCT374DW
U36
SN74HCT374DW
U37
U39
16L8
LBUS STATE MACHINE
P12
VCC
U41
SN74HC85AD
S7
R26 10K
R27 10K
R28 10K
R29 10K
ONBOARD
TITLE SCHEMATIC, REGISTER BASED BREADBOARD
SIZE C
CAGE CODE
DWG NO. 50-0026-000
REV. A
SCALE NONE
FILE 0026-000.S06
SHEET 17 OF 19



TITLE: SCHEMATIC, REGISTER BASED BREADBOARD				
SIZE: C	CAGE CODE:	DWG NO: 50-0026-000		REV: A
SCALE: NONE	FILE: 0026-000.S06		SHEET: 18 OF 19	

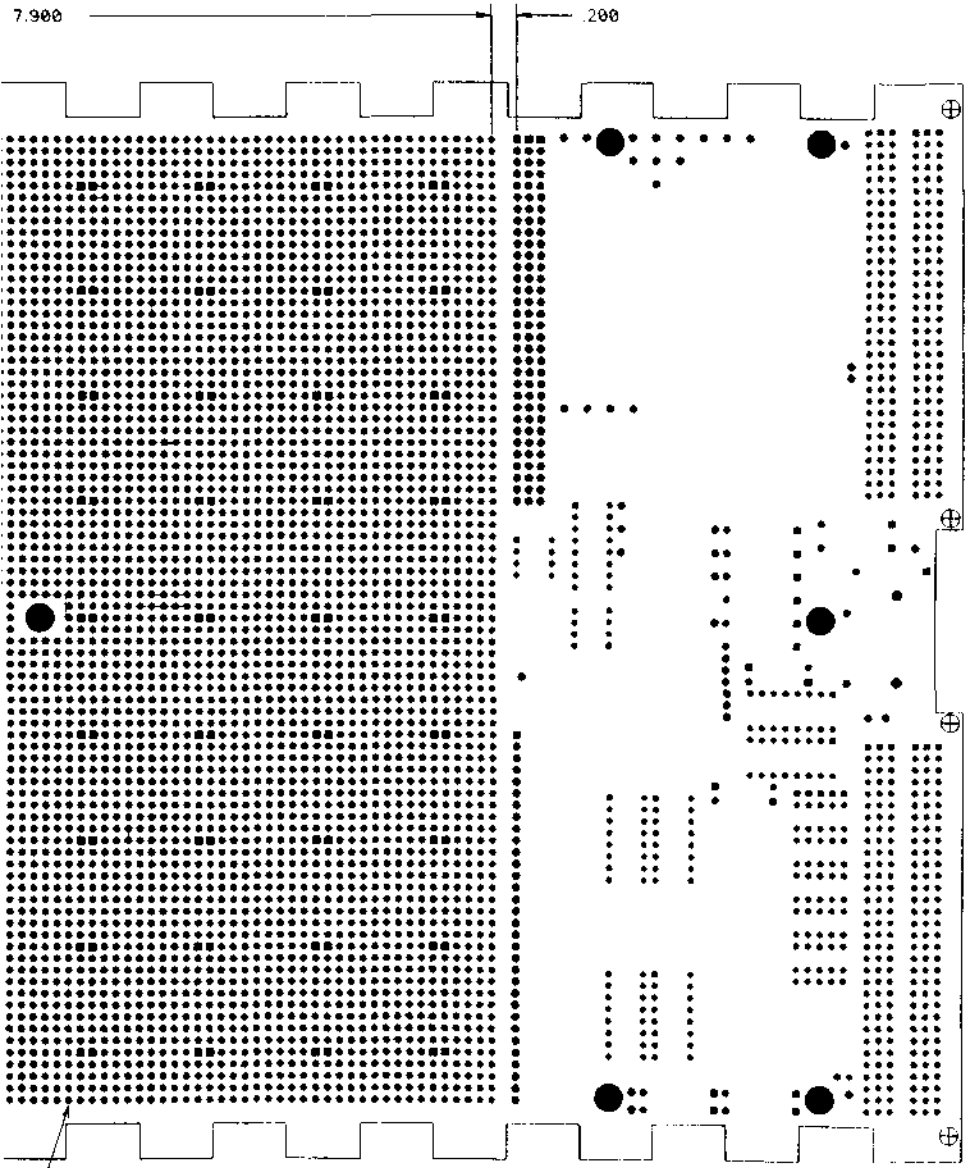


TITLE: SCHEMATIC, REGISTER BASED BREADBOARD				
SIZE: C	CAGE CODE	DWG NO: 50-0026-000		REV: A
SCALE: NONE	FILE: 0026-000.S06	SHEET: 19 OF 19		



HOLE SPACING IS .1 INCH VERTICALLY
ROWS ARE SPACED 1 INCH APART
ALTERNATING COLUMNS ARE OFFSET BY .050 INCH
PADS ARE CONNECTED 1 FOR 1 TO THE ADJACENT
CONNECTOR PATTERN, TYPICAL 2 PLACES

NOTE REFERENCE THE STANDARD C SIZE PCB OUTLINE DRAWING
FOR FURTHER MECHANICAL DETAIL



HOLE PATTERN IS .1 BY .1 INCH THROUGH OUT
THE SQUARE PADS ARE TIED TO THE POWER PLANES
THE LEFT PAD IN EACH PAIR IS VCC
WHILE THE RIGHT PAD TIES TO GROUND

Title	PCB, REGISTER BASED BREADBOARD MECHANICAL OUTLINE FOR HOLE PLACEMENT		
Size C	Number REFERENCE DRAWING		Rev A
Date	1 28 92	Drawn by	AJG
Filename		Sheet	1 of 1

SECTION 8 PARTS LISTS

70-0010-000	Final Assembly	8-2
52-0010-000	PCB Assembly, "C" Size Breadboard	8-3
52-0011-000	PCB Assembly, Message-Based Interface	8-4
52-0026-000	PCB Assembly, Register-Based Breadboard	8-6

70-0010-000, FINAL ASSEMBLY, 7064

REF. DESIG.	RACAL PART NO.	DESCRIPTION
{1} 2	11-0010-001	LED, PANEL MOUNT, .156 DIA, RED
{2} 17	37-0010-025	SCREW, 4-40 PAN HEAD, SEMS
{3} 4	37-0011-062	SCREW, 4-40 PAN HEAD
{4} 4	37-0012-004	WASHER, FLAT, 4-40
{5} 4	37-0013-004	WASHER, SPLIT LOCK, 4-40
{6} 6	37-0016-018	SCREW, 2-56 FLAT HEAD
{7} 1	37-0017-000	VXI EXTRACTOR HANDLE, TOP, GREY
{8} 1	37-0017-001	VXI EXTRACTOR HANDLE, BOTTOM, GREY
{9} 1	37-0017-002	VXI EXTRACTOR HARDWARE KIT
{10} 1	41-0010-001	FRONT PANEL, C SIZE BREADBOARD SINGLE
{11} 1	41-0010-002	BASE PLATE, C SIZE BREADBOARD SINGLE
{12} 1	41-0010-003	COVER PLATE, C SIZE BREADBOARD SINGLE
{13} 1	43-0010-000	LABEL, STATIC WARNING
{14} 1	43-0011-000	LABEL, RDII SERIAL/MODEL NUMBER
{15} 1	52-0010-000	ASSY., PCB, VXI "C" SIZE BREADBOARD
{16} 1	52-0011-000	ASSY, PCB VXI INTERFACE MODULE 8 MHz
{17} 2	52-0013-095	ASSY., CABLE LED

70-0010-001, FINAL ASSEMBLY, 7064-2

REF. DESIG.	RACAL PART NO.	DESCRIPTION
{1} 2	11-0010-001	LED, PANEL MOUNT, .156 DIA, RED
{2} 17	37-0010-025	SCREW, 4-40 PAN HEAD, SEMS
{3} 4	37-0011-062	SCREW, 4-40 PAN HEAD
{4} 4	37-0012-004	WASHER, FLAT, 4-40
{5} 4	37-0013-004	WASHER, SPLIT LOCK, 4-40
{6} 6	37-0016-018	SCREW, 2-56 FLAT HEAD
{7} 1	37-0017-000	VXI EXTRACTOR HANDLE, TOP, GREY
{8} 1	37-0017-001	VXI EXTRACTOR HANDLE, BOTTOM, GREY
{9} 1	37-0017-002	VXI EXTRACTOR HARDWARE KIT
{10} 1	41-0010-001	FRONT PANEL, C SIZE BREADBOARD SINGLE
{11} 1	41-0010-002	BASE PLATE, C SIZE BREADBOARD SINGLE
{12} 1	41-0010-003	COVER PLATE, C SIZE BREADBOARD SINGLE
{13} 1	43-0010-000	LABEL, STATIC WARNING
{14} 1	43-0011-000	LABEL, RDII SERIAL/MODEL NUMBER
{15} 1	52-0010-000	ASSY., PCB, VXI "C" SIZE BREADBOARD
{16} 1	52-0011-001	ASSY, PCB VXI INTERFACE MODULE 12 MHz
{17} 2	52-0013-095	ASSY., CABLE LED

52-0010-000, PCB ASSY., "C" SIZE BREADBOARD

REF. DESIG.	RACAL PART NO.	DESCRIPTION
C1-8	05-0010-103	CAP, MONO, 0.01 UF, 50V, SMD 1206
C9	05-0011-336	CAP, TANTA, 3.3UF, 16V, RADIAL
C10-32	05-0010-103	CAP, MONO, 0.01UF, 50V, SMD 1206
F1	35-0010-050	FUSE, 5A, AXIAL
F2,3	35-0010-020	FUSE, 1A, AXIAL
F4,5	35-0010-050	FUSE, 5A, AXIAL
F6,7	35-0010-020	FUSE, 1A, AXIAL
J101,102	27-0012-032	CONNECTOR, SOCKET STRIP, PIN
L1	09-0010-430	IND, POWER, RADIAL, 4.3UH, 5.5A
L2-8	09-0011-000	IND, BEAD, AXIAL
P1,2	27-0010-096	CONNECTOR DIN RT. ANGLE 96-PIN MALE
R1-40	01-0010-103	RES, CHIP 10K, 0.12W 5PCT, SMD 1206
SW1	29-0010-004	SWITCH, DIP, 4 POS., LOW PROFILE
U1-4	17-0012-652	IC, DIGITAL, 74HCT652, S024
U5	17-0012-000	IC, DIGITAL, 74HCTOO, S014
U6	17-0017-000	IC, DIGITAL, 74HCA078, S014
U7	17-0011-139	IC, DIGITAL, 74F139, S016
U8-10	17-0011-138	IC, DIGITAL, 74F138, S016
U11	17-0011-138	IC, DIGITAL, 74F138, S016
U12	17-0012-085	IC, DIGITAL, 74HCT85, S016
U13	17-0011-245	IC, DIGITAL, 74F245 S020
U14,15	17-0012-374	IC, DIGITAL, 74HCT374, S020
U16	17-0011-004	IC, DIGITAL, 74F04, S014
U17	17-0011-032	IC, DIGITAL, 74F32, S014
U18-25	17-0012-652	IC, DIGITAL, 74HCT652, S024
U26,U27	17-0012-000	IC, DIGITAL, 74HCT00, S014
Z1	01-0018-332	RES NETWORK, 3.3K, 8 RES, SMD
{19}1	33-0010-000	PCB VXI "C" BREADBOARD (UNLOADED)

52-0011-000, PCB ASSY., VXI MESSAGE BASED INTERFACE, 8 MHz

REF. DESIG.	RACAL PART NO.	DESCRIPTION
C1	05-0010-103	CAP, MONO, 0.01 UF, 50V, SMD 1206
C2	05-0012-336	CAP, TANTA, 3.3 UF, 16V, SMD
C3-C27	05-0010-103	CAP, MONO, 0.01 UF, 50V, SMD 1206
C28	05-0012-336	CAP, TANTA, 3.3UF, 16V, SMD
P1	27-0013-004	CONNECTOR, RIGHT ANGLE, 4 PINS
P100	27-0013-064	CONNECTOR, RIGHT ANGLE, 64 PINS
P101A,B,C; P102A,C	27-0011-032	CONNECTOR, PIN STRIP, 32 PINS
R1-4	01-0011-103	RES, CHIP, 10K, 0.06W, 5PCT, SMD 0805
R5,6	01-0011-102	RES, CHIP, 1K, 0.06W, 5PCT, SMD 0805
R7-16	01-0011-472	RES, CHIP, 4.7K, 0.06W, 5PCT, SMD 0805
R19	01-0011-102	RES, CHIP, 1K, .06W, 5PCT, SMD 0805
R20	01-0011-471	RES, CHIP, 470 OHM, 0.06W, 5PCT, SMD 0805
R21	01-0011-102	RES, CHIP, 1K, 0.06W, 5PCT, SMD 0805
R22	01-0011-472	RES, CHIP, 4.7K, 0.06W, 5PCT, SMD 0805
R23,24	01-0011-331	RES, CHIP, 330 OHM, 0.06W, 5PCT SMD 0805
R25	01-0011-104	RES, CHIP, 100K, 0.6W, 5PCT, SMD 0805
R26,27	01-0011-102	RES, CHIP, 1K, 0.06W, 5PCT, SMD 0805
R28	01-0011-472	RES, CHIP, 4.7K, 0.06W, 5PCT SMD 0805
R29	01-0013-825	RES, CHIP, 825 OHM, 0.06W 1PCT, SMD 0805
R30-37	01-0011-472	RES, CHIP, 4.7K 0.06W, 5PCT SMD 0805
R38-41	01-0011-102	RES, CHIP, 1K, 0.06W, 5PCT SMD 0805
R42-51	01-0011-220	RES, CHIP, 22 OHM, 0.06W, 5PCT SMD 0805
R52	01-0011-331	RES, CHIP, 330 OHM, 0.06W, 5PCT SMD 0805
R53	01-0011-471	RES, CHIP, 470 OHM, 0.06W, 5PCT SMD 0805
R56,R57	01-0011-472	RES, CHIP, 4.7K, 0.06W, 5PCT SMD 0805
SW1	29-0011-008	SWITCH, DIP, 8 POS., SMD
U1	19-0010-001	IC, MEMORY, EPROM, M5M27C102JK-15
U2,3	19-0011-000	IC, MEMORY, RAM, KM62256ALG-10
U4	17-0013-032	IC, DIGITAL, 74HC32, S014
U5,6	19-0013-000	IC, MEMORY EEPROM, NQ28C64-200
U7	17-0011-074	IC, DIGITAL, 74F74, S014
U8	17-0014-038	IC, DIGITAL, 74S38, S014
U9	17-0011-138	IC, DIGITAL, 74F138, S016
U10	17-0015-000	IC, MICROPROCESSOR, MC68000FN-8
U11	17-0012-008	IC, DIGITAL, 74HCT08, S014
U12	21-0013-001	IC, PAL, 20L8AFN-25
U14	17-0010-148	IC, DIGITAL, 74LS148, S016
U15	17-0012-151	IC, DIGITAL, 74HCT151, S016
U16,17	17-0012-138	IC, DIGITAL, 74HCT138, S016
U18	17-0012-688	IC, DIGITAL, 74HCT688, S020
U19	17-0016-000	IC, TIMER, MC68230FN-8
U20	17-0012-074	IC, DIGITAL, 74HCT74, S014
U21,22	17-0011-641	IC, DIGITAL, 74F641, S020
U23	15-0010-005	IC, INT., TL7705, S08
U24	17-0012-032	IC, DIGITAL, 74HCT32, S014
U25	17-0012-244	IC, DIGITAL, 74HCT244, S020
U26	17-0012-688	IC, DIGITAL, 74HCT688, S020
U28,29	19-0014-000	IC, MEMORY, DUAL PORT RAM, IDT7130SAJ100
U30	17-0012-374	IC, DIGITAL, 74HCT374, S020

52-0011-000 CONTINUED

REF. DESIG.	RACAL PART NO.	DESCRIPTION
U31	17-0012-074	IC, DIGITAL, 74HCT74, S014
U32,33	17-0012-367	IC, DIGITAL, 74HCT367, S016
U34	17-0011-138	IC, DIGITAL, 74F138, S016
U35	17-0011-000	IC, DIGITAL, 74F00, S014
U36	21-0010-001	IC, PAL, 22V10FN-25
U37,38	17-0011-652	IC, DIGITAL, 74F652, S024
U39	17-0011-032	IC, DIGITAL, 74F32, S014
U40	21-0011-001	IC, PAL, 16L8AFN-15
U41	17-0012-374	IC, DIGITAL, 74HCT374, S020
U42	17-0012-030	IC, DIGITAL, 74HCT30, S014
U43,44	17-0012-244	IC, DIGITAL, 74HCT244, S020
U45	21-0012-001	IC, PAL, 16R4AFN-15
U46	17-0011-367	IC, DIGITAL, 74F367, S016
U47	17-0011-245	IC, DIGITAL, 74F245, S020
Z1	01-0018-102	RES NETWORK, 1K, 2PCT
{45}1	27-0014-044	SOCKET, PLCC, 44-PIN
{46}1	27-0014-068	SOCKET, PLCC, 68-PIN
{48}1	33-0011-000	PCB, VXI INTERFACE MODULE (UNLOADED)
{49}4	37-0014-025	SPACER, SWAGE, 4-40 X .250, 3/16 OD

REGISTER BASED BREADBOARD PARTS LIST

Quantity	Type	Value	Ref Designators
4	16L8		U12,U15,U24,U39
1	96_PIN_EURO_CONN		P11
1	BPAL20L8-25CFN		U28
2	BPAL22V10CFN		U25,U26
14	CAP	.01 uF	C1,C2,C4,C5,C6,C7,C8,C9, C10,C11,C12,C13,C14,C15
10	CAP	.01uF	C21,C22,C23,C24,C25,C26, C27,C28,C29,C30
1	CAP	3.3uF	C3
1	CONN4PIN		P10
5	CONN_12PIN		P5,P6,P7,P8,P9
1	CONN_32_PIN		P12
4	DIODE	HP2811	D1,D2,D3,D4
2	FUSE	10A	F1,F6
4	FUSE	2A	F2,F3,F4,F5
1	FUSE	5A	F7
2	HRES	330	R10,R12
8	HRES	4.7K	R2,R3,R5,R7,R11,R30,R31, R32
1	IND	5.0uH	L2
7	IND	BEAD	L1,L3,L4,L5,L6,L7,L8
1	JUMPER		JP1
1	LM555CMA		U35
1	MC10H116		U40
1	MC10H123		U38
4	RES16DIPI	1K	R6,R8,R9,R13
1	RES16DIPI	3.3K	R4
2	SCAP	.001uF	C18,C20
1	SCAP	.01uF	C17
2	SCAP	.1uF	C16,C19
5	SN74F244DW		U1,U4,U7,U8,U11
3	SN74F245DW		U3,U5,U34
1	SN74F641DW		U31
4	SN74F652DW		U2,U6,U10,U13
1	SN74HC85AD		U41
1	SN74HCT02D		U20
1	SN74HCT04D		U22
1	SN74HCT08D		U33
1	SN74HCT138D		U30
6	SN74HCT244DW		U9,U16,U18,U21,U23,U42
1	SN74HCT273DW		U17
1	SN74HCT32D		U27
2	SN74HCT374DW		U36,U37
2	SN74HCT688DW		U14,U29

REGISTER BASED BREADBOARD PARTS LIST continued

Quantity	Type	Value	Ref Designators
1	SN74HCT74D		U32
1	SN74S38D		U19
33	SRES	4.7K	R33,R34,R35,R36,R37,R38, R39,R40,R41,R42,R43,R44, R45,R46,R47,R48,R49,R50, R51,R52,R53,R54,R55,R56, R57,R58,R59,R60,R61,R62, R63,R64,R65
5	SWITCH8DIP	VAL	S1,S2,S3,S4,S5
2	SWITCH_4DIP		S6,S7
40	TESTPOINT		E1,E2,E3,E4,E5,E6,E7,E8, E9,E10,E11,E12,E13,E14, E15,E16,E17,E18,E19,E20, E21,E22,E23,E24,E25,E26, E27,E28,E29,E30,E31,E32, E33,E34,E35,E36,E37,E38, E39,E40
2	VME_P1		P1,P3
1	VRES	100K	R21
4	VRES	10K	R26,R27,R28,R29
2	VRES	180	R23,R25
2	VRES	270	R22,R24
7	VRES	4.7K	R14,R15,R16,R17,R18,R19, R20
1	VRES	825	R1
2	VXI_P2		P2,P4

Total Parts: 216

APPENDIX A

SOFTWARE COMMAND LIBRARY

The software command library available on the 7064 has commands accessible either through the message based interface directly or only during firmware development. When defining these commands in this section the user will be told wether these commands are available only through the message based interface, only during firmware development, or through both.

Port Commands

cinbit (uint card, uint port, uint bitnum);
cinport (uint card, uint cport);
cinportb (uint card, uint cport);
coutbit (uint card, uint cport, uint bit_number, uint value);
coutport (uint card, uint cport, uint value);
coutportb (uint card, uint cport, uint value);
cpulsehi (uint card, uint cport, uint bit_number);
cpulselo (uint card, uint cport, uint bit_number);
crstbit (uint card, uint cport, uint bit_number);
csetbit (uint card, uint cport, uint bit_number);
inbit (uint port, uint bit_number);
inport (uint port);
inportb (uint port);
outbit (uint port, uint bit_number, uint value);
outport (uint port, uint value);
outportb (uint port, uint value);
pulsehi (uint port, uint bit_number);
pulselo (uint port, uint bit_number);
rstbit (uint port, uint bit_number);
setbit (uint port, uint bit_number);

These commands are used to talk to the I/O lines and are described in detail in Section 3 of this manual. Examples are also given.

Intermetrics C-cross Assembler Commands*

```
abs (int i);                    strcat (char *dest, char *src);
atoi (char *string);            strchr (char *src, char c);
atol (char *string);            strcmp (char *src1, char *src2);
index (char *src, char c);      strcpy (char *dest, char *src);
isalnum (int c);                strcspn (char *src, char *list);
isalpha (int c);                stringcmpi (char *src1, char *src2);
iscntrl (int c);                strlen (char *src);
isdigit (int c);                strncat (char * dest, char * source, uint max);
isgraph (int c);                strncmp (char *src1, char *src2, int num);
islower (int c);                strncpy (char *dest, char *src, int num);
isprint (int c);                strpbrk (char *src, char *list);
ispunct (int c);                strrchr (char *src, char c);
isspace (int c);                strspn (char *src, char *cmp);
isupper (int c);                strtok (char *src, char *sep);
isxdigit (int c);               swab (char *src, char *dest, int nbytes);
labs (long i);                  tolower (int c);        toupper (int c);
memccpy (char *dest, char *src, char c, int num);
memchr (char *src, char c, int num);
memcmp (char *src1, char *src2, int num);
memcpy (char *dest, char *src, int num);
memset (char *dest, char c, int num);
rand ();
rcopy (void *table);
rindex (char *src, char c);
sprintf (char *buffer, char *format, ...);
srand (int seed);
sscanf (char *buffer, char *format, ...);
```

These commands are defined in detail in the Intermetrics manuals that are supplied with the C software and debugger manuals. These are not required if the user is not writing his own firmware.

Non-Vol Management

delete_nv (void *nv_blkptr); *
This deletes a structured non-volatile record.

> **Example:** delete_nv(addr); - deletes the record at (addr).

locate_nv (int rec_type, int sequence); *
Searches thru non-vol to find a structured record.

> **Example:** addr=locate_nv(2,1); - locates nv record type 2, first occurance (addr).

nvcheckport(unsigned int port, unsigned int mask, unsigned int value);
Put a record in the "record" area that tells the system to read a particular **port**, AND it with the **mask** and compare it with **value**. If they don't compare, report the mismatch. This could be used to make a system note if all required boards were not present.

> **Example:** nvcheckport(0,#HF,#H0); - Looks at the lower 4 bits of port 0 and checks to see if a 0000 occurs. If it doesn't an error is generated.

nvnocheck(void);
Remove all check records (generated using **nvcheckport)** from non-vol memory. Used for starting over when assigning check values or if checking is no longer desired.

> **Example:** nvnocheck(); - Removes all check records.

nvsetflag(unsigned int flag);*
Sets an individual flag. Two flags are currently supported, 0 and 1. The user can set nvsetflag instead of using nvsetvect. These are described in more detail in the program nv.h on the accompanying disk.

> **Example 1:** nvsetflag(1); - sets flag ROM_INIT. Upon power up an instruction is stored in memory location #H10000, telling the firmware to go to the user initialization routine.
> **Example 2:** nvsetflag(0); - sets flag ROM_VECTOR. Upon power up the firmware uses the information stored in the following locationsl
> #H10000 d.c.I Parser
> #H10004 d.c.I Initialization
> #H10008 d.c.I Clear
> #H1000C d.c.I Error

nvnoflags(void); *
Removes all flags (generated using nvsetflag) from non-vol memory.

nvsetport(unsigned int port, unsigned int value);
Put a record in the "record" area that tells the firmware to initialize a particular port. At the end of the power-up sequence, **value** is stored in **port**.

Example: nvsetport(01,#HF); - sets port 01 to value #HF at power on.

nvnoset(void);
Remove all set port records (generated using **nvsetport**) from non-vol memory. Used when an initial value different from the reset value of a port is no longer desired.

Example: nvnoset(); - removes the values set for all ports in non-vol.

nvsetvxireg(unsigned int register_num, unsigned int value);
Put a record in non-vol that will cause **register_num** of the VXI registers to get initialized to **value** instead of it's regular value. This might be used to customize something like the manufacturer's ID or the model number. Note: improper usage of this command may cause a violation of the VXIbus standard. Use with caution.

Example: nvsetvxireg(0,1011111111111011); - Stores the following information into the VXIbus registers: message based device, A16 only, and sets the manufacturers ID code to 4091.
Please consult VXIbus specification revision 1.3, page 106 before using.

nvnovxireg(void);
Remove all records from non-vol that would be used to initialize VXI registers (generated by **nvsetvxireg**).

nvsetvect(int vector_num, void(*isr)());
Store an address of routine **isr** in **vector_num** of the system block's vectors. These vectors are used during power-up to setup interrupt vectors, etc. An address of zero causes the vector to be ignored.

Example: nvsetvect(26,#H10046); - stores the address (#H10046) of the routine in system block vector 26.

nvsizeglobal(unsigned long size);
Store **size** in the system block for the amount of RAM to be used for global variables. This would be used if the default value was not large enough. During power-up this value is used to set the size of the user global area. A value of zero causes the default value to be used. The default value is 4K bytes. The size is specified in bytes.

Example: nvsizeglobal(#H2000); - Sets 8K of RAM to be used

nvsizeheap(unsigned long size);
Store **size** in the system block for the amount of RAM to be used for the heap. This would be used if the default value was not large enough. During power-up this value is used to set the size of the heap area. A value of zero causes the default value to be used. The default value is 4K bytes. The size is specified in bytes.

Example: nvsizeheap(#H2000); - sets heap size to 8K.

nvsizenvcode(unsigned long size);
Store **size** in the system block for the amount of non-vol to be set aside for unstructured use. This would be used if the default value was not large enough. During power-up this value is used to set the size of the unstructured non-vol area. No "records" can be stored in the unstructured area. A value of zero causes the default value to be used. The default value is 0 bytes (all available nonvol is used for "records"). The size is specified in bytes. **Please Note:** This can not be reduced once set, without resetting non-vol.

Example: **nvsizeenvcode(#H1000); - sets size of block to 4K.**

nvsizeoutq(unsigned long size);
Store **size** in the system block for the amount of RAM to be used for the output queue. This would be used if the default value was not large enough. During power-up this value is used to set the size of the output queue. A value of zero causes the default value to be used. The default value is 512 words. The size is specified in words, one word for each character.

Example: nvsizeoutq(2000); - Sets output queue to be 2000 words

nvsizeinq(unsigned long size);
Store **size** in the system block for the amount of RAM to be used for the input queue. This would be used if the default value was not large enough. During power-up this value is used to set the size of the input queue. A value of zero causes the default value to be used. The default value is 170 words. The size is specified in words, one word for each character.

Example: nvsizeinq(200); - Sets input queue to be 200 words

nvsizesecq(unsigned long size);
Store **size** in the system block for the amount of RAM to be used for the secondary queue. This would be used if the default value was not large enough. During power-up this value is used to set the size of the secondary queue. A value of zero causes the default value to be used. The default value is 40 words. The size is specified in words, one word for each character.

Example: **nvsizesecq(200); - sets size to 200 words.**

nvstartinuser(unsigned int yes);
Store **yes** in the system block for whether to start in the user's parser or the standard parser. This would be used if there was a dedicated user parser and the default command set was not desired. During power-up this value is used to set which parser is active. A value of zero causes the default parser to be used. A non-zero value causes the user parser to be used. The user parser address is one of the vectors that can be stored in non-vol (see **nvsetvect**). Storing a non-zero **yes** is the same as having the commander send the command to switch parsers.

Example: nvstartinuser(1); - Enables nvsetvect

nvstorecode(void *source, void *dest, unsigned long length);
Move a block of information from **source** to **dest** where **dest** is an address in the unstructured area of non-vol. The length of the block is **length** bytes. This could be used to copy a user parser that had been previously loaded into RAM into the non-volatile area.

Example: nvstorecode(#H403000,#H602000,#H2000); - Stores an S-record of 8 Kbytes (#H2000) loaded from RAM at address #H403000 into non-vol at address #H602000.

retrieve_nv (void *dest, void *nv_blkptr); *

Retrieves information out of structured non-vol and puts it into user data area.

> **Example:** addr=locate_nv(2,1); followed by retrieve_nv(mybuf,addr); - retreives the information from *addr*, and stores it into mybuf in RAM location.

store_nv (int rec_type, char *dscrpt, void *dataptr, long size); *

Stores information from user data area into structured non-vol.

> **Example:** store_nv(2,"myinfo",mybuf,(long)size of mybuf);

Memory Management

vtl_calloc (long count, long size); or vtl_malloc (long size); *

Allocates a block of memory in the heap.

> **Example: addr=vtl_calloc(4,200); - allocates 4 pieces of memory of 200 bytes each.**

vtl_cfree(void *block); or vtl_free (void *block); *

Frees a block of memory in the heap.

> **Example:** vtl_free(addr);

peek (int * address);

Retrieves a word from specified memory location.

peekb (char * address);

Retrieves a byte from specified memory location.

poke (int * address, int value);

Puts a word into specified memory location.

pokeb (char * address, char value);

Puts a byte into specified memory location.

> **Examples:** Examples are given in section 4.4.

Primary Input Queue Management *

eost_pri (void);
If non-zero returned, then last byte fetched from the primary queue had an end bit.

> **Example:** i=eost_pri(); - returns a zero unless last byte fetched has end bit set.

getchar_pri (void);
This command gets an 8-bit character out of the primary queue.

> **Example:** i=getchar_pri();

getchar_raw_pri(void);
This character gets a 16-bit character out of the primary queue.

> **Example:** i=getchar_raw_pri();

in_stat_pri (void);
Returns the number of characters in the primary input queue.

> **Example:** i=in_stat_pri();

ungetchar_pri(int c);
Disposes of the character after the *getchar* command.

> **Example:** ungetchar_pri(i);

Scondary Input Queue Management *

eost (void);
If non-zero returned, then last byte fetched from the secondary queue had an end bit.

> **Example:** i=eost(); - returns a zero unless last byte fetched has end bit set.

getchar_raw (void);
This character gets a 16-bit character out of the secondary queue.

Example: i=getchar_raw();

in_stat (void);
Returns the number of characters in the secondary input queue.

Example: i=in_stat();

putchar_sec(int c);
Puts a character into the secondary queue.

Example: putchar_sec(i);

vt1_getchar (void);
Gets a character out of the secondary input queue.

Example: i=vt1_getchar();

vt1_ungetc (int ch);
Disposes of the character after the *vt1_getchar* command.

Example: vt1_ungetc(i);

Interrupt management

disable (void); *
This command disables interrupts.

enable (void); *
This command enables interrupts.

Note: During initiallization the interrupts are off, and during normal operation the interrupts are on.

event (uint event_value); *
Generates an interrupt on the backplane (value of interrupt, as defined in the VXIbus specifications).

Example: event(#H7F);

getvect(int interruptno);
Retrieves a vector.

Example: addr=getvect(26);

restoreintlevel (uint old_level); *
Restores the interrrupt level.

Example: restoreintlevel(i);

setintlevel (uint new_level); *
Sets interrupt level.

Example: i=setintlevel(7);

setvect (int interruptno, void (*isr) ());
Puts a vector into RAM location.

example: setvect(26,myroutine);

Output Management

base (uint base_value);
sets the base of the output. Defined in detail in section 3.5.1.

out_stat (void); *
Returns the number of characters in the output queue.

Example: i=out-stat();

vt1_out(uint value, uint end); .
Converts a number (dependant on *base*) into ascii, and puts it into the output queue.

Example: vt1_out(i,1);

vt1_outl(ulong value, uint end);*
Converts a number (dependant on *base*) into ascii, and puts it into the output queue.

Example: vt1_out(#H800000,1);

vt1_putchar (uint ch, uint end); *
Puts a character into the output queue.

Example: vt1_putchar(i,1);

vt1_puts (char *s, uint end); *
Puts a complete string into the output queue.

Example: vt1_puts("myinfo",1);

Timer Management

delay (ulong microseconds); *
This allows the user to set a delay in his program. This will insert a minimum delay of microseconds.

gettime (struct time *timep); *
Puts the current time into the structure address.

Example: struct time mytime;
gettime(mytime);

gettimel (void);
Returns the number of microseconds since power on (modulo 2^{32})

raw_time * point_time (void); *
Returns the time in raw data.

settime (struct time *timep); *
Sets up the current time.

sleep (uint seconds); *
Sets a delay for seconds.

Triggering

triggen(void);
triggin (uint line); /* 0:7 is line, 8 is disable */
trigout (uint line); /* 0:7 is line, 8 is disable */

These commands are described in detail in section 3.5.5

Misscelaneous commands

clear_done (void); *
The user should send this command to his routine in response to the WS clear command (VXIbus specifications).

setparser (ulong address);
Sets the user parser.

NOTE: The commands marked with an asterix (*) are only available for firmware development. All other commands are available either through the message based interface or for firmware development.

Production Rules Describing 7064 Command Syntax.

```
<cmdlist> => <cmd>+    /* A 7064 command list is 1 or more 7064 commands */

/* A 7064 command "is a" bit command "or a" byte command, ... */
<cmd> => <bit_cmd> ';'
      | <byte_cmd> ';'
      | <word_cmd> ';'
      | <trig_cmd> ';'
      | <block_cmd> ';'
      | <base_cmd> ';'
      | <usr_cmd> ';'
      | <peek_cmd> ';'
      | <poke_cmd> ';'
      | <nv_cmd> ';'
      | <vect_cmd> ';'
      | <setparser_cmd> ';'
      | <parsemode_cmd> ';'
```

```
/* A 7064 <bit_cmd> "is a" SETBIT command "or a" RSTBIT command, ... */
<bit_cmd> => SETBIT  '(' <port> ',' <bitnum> ')'
          | RSTBIT  '(' <port> ',' <bitnum> ')'
          | OUTBIT  '(' <port> ',' <bitnum> ','<bit_value>')'
          | INBIT   '(' <port> ',' <bitnum> ')'
          | CSETBIT  '(' <card> ',' <port> ',' <bitnum> ')'
          | CRSTBIT  '(' <card> ',' <port> ',' <bitnum> ')'
          | COUTBIT  '(' <card> ',' <port> ',' <bitnum> ',' <bit_value> ')'
          | CINBIT   '(' <card> ',' <port> ',' <bitnum> ')'
          | PULSEHI  '(' <port> ',' <bitnum> ')'
          | PULSELO  '(' <port> ',' <bitnum> ')'
          | CPULSEHI '(' <card> ',' <port> ',' <bitnum> ')'
          | CPULSELO '(' <card> ',' <port> ',' <bitnum> ')'

/* 7064 byte-level commands */
<byte_cmd> => OUTPORTB  '(' <port> ',' <byte> ')'
           | INPORTB   '(' <port> ')'
           | COUTPORTB '(' <card> ',' <port> ',' <byte> ')'
           | CINPORTB  '(' <card> ',' <port> ')'

/* 7064 word-level commands */
<word_cmd> => OUTPORT  '(' <port> ',' <word> ')'
           | INPORT   '(' <port> ')'
           | COUTPORT '(' <card> ',' <port> ',' <word> ')'
           | CINPORT  '(' <card> ',' <port> ')'

/* 7064 trigger commands */
<trig_cmd> => TRIGGIN '(' <trig_line> ')'
           | TRIGOUT '(' <trig_line> ')'
           | TRIGGEN '(' ')'

/* Download user-written code into the 7064 for later execution */
<block_cmd> => DOWNLOAD <func_name> '(' ')' ';' {<S-record><terminator>}+

/* Sets format for subsequent 7064 numerical output and misc */
<base_cmd> => BASE '(' <basevalue> ')'
             | GETTIMEL '(' ')'
```

```
/* Invokes a user-supplied function that's been previously downloaded */
<usr_cmd> => <func_name> '(' ')'

<peek_cmd> => PEEK  '(' <address> ')'   /* Peek word */
        | PEEKB '(' <address> ')'   /* Peek byte */

<poke_cmd> => POKE  '(' <address> ',' <word> ')'  /* poke a word */
        | POKEB '(' <address> ',' <byte> ')'  /* poke a byte */

<vect_cmd> => SETVECT '(' <word> ',' <address> ')'
            | GETVECT '(' <word> ')'

<nv_cmd>   => NVCHECKPORT '(' <port> ',' <byte> ',' <mask> ')'
            | NVNOCHECK '(' ')'
            | NVSETPORT '(' <port> ',' <byte> ')'
            | NVNOSET '(' ')'
            | NVSETVXIREG '(' <reg> ',' <word> ')'
            | NVNOVXIREG '(' ')'
            | NVSETVECT '(' <vector> ',' <address> ')'
            | NVSIZEGLOBAL '(' <length> ')'
            | NVSIZEHEAP '(' <length> ')'
            | NVSIZEINQ '(' <length> ')'
            | NVSIZENVCODE '(' <length> ')'
            | NVSIZEOUTQ '(' <length> ')'
            | NVSIZESECQ '(' <length> ')'
            | NVSTARTINUSER '(' <word> ')'
            | NVSTORECODE '(' <address> ',' <address> ',' <length> ')'

/* Sets an alternate user parser */
<setparser_cmd> => SETPARSER '(' <address> ')' ';'

<parsemode_cmd> => _ _USR_ _          /* switch to user parser mode */
            | _ _VTI_ _     /* switch to 7064  parser mode */

<func_name> => /* An arbitrarily-long, case-insensitive, C-style identifier */

<S-record> => /* A Motorola S-record. 7064 supports S0,S1,S2,S3,S8,S9 types */

<terminator> => { '\r' | '\n' }+ | EOI asserted
```

<bit_number> => <unsigned integer const> of range [0 .. 7], inclusive
<trig_line> => <unsigned integer const> of range [0 .. 8], inclusive
<port> => <unsigned integer const> of range [0 .. 11], inclusive
<byte> => <unsigned integer const> of range [0 .. 2^8 - 1], inclusive
<word> => <unsigned integer const> of range [0 .. 2^{16} - 1], inclusive
<bit_value> => 0 | 1
<base_value> => 2 | 8 | 10 | 16
<address> => <unsigned integer const> of range [0 .. 2^{32} - 1], inclusive
<mask> => <unsigned integer const> of range [0 .. 2^8 - 1], inclusive
<reg> => <unsigned integer const> of range [0 .. 31], inclusive
<vector> => <unsigned integer const> of range [0 .. 63], inclusive
<length> => <unsigned integer const> of range [0 .. 2^{31} - 1], inclusive

<unsigned integer const> =>
The following regular expressions describe the numerical formats recognized by the 7064 command parser:

- IEEE Hex : #[hH][a-fA-F0-9]+
- IEEE Octal : #[qQ][0-7]+
- IEEE Binary : #[bB][01]+
  - C-style Hex : 0[xX][a-fA-F0-9]+
  - C-style Octal : 0[0-7]*
  - Decimal : [0-9]+

NOTES:

1. In general, the 7064 command parser accepts a 'free-format' syntax. That is, the placement of new lines and whitespace isn't critical. There are three exceptions:

   a. The "USR_SCAN();" command should appear alone on a terminated line.

   b. The "VXI_SCAN();" command should appear alone on a terminated line.

   c. In S-record download, each Motorola S-record should appear alone on a terminated line.

2. All identifiers are case-insensitive. That is, "INPORT" is the same as "inport" and "Inport", etc.

APPENDIX B

7064 COMMAND SYNTAX AND SAMPLE COMMAND STRINGS

1.0 Introduction

All commands must be terminated by a semicolon ';' except for the download command. Users may download a named block of object code (encoded in Motorola S-record format) into the 7064 and later invoke that block of code as if making a C function call (just like any of the other built-in commands). Argument lists for user-supplied functions are not currently supported.

2.0 Formal Defination

The 7064 command set may be concisely described by a simple"grammar" and a set of "regular expressions".

2.1 Regular Expressions

Regular expressions describe text patterns by a string of symbols tied together with the following operators:

(a) **Concatenation.**

If two characters are adjacent in a pattern, then there is a match if and only if there is a match in the source text. For example, AB means A followed by B.

(b) **Character sets.**

Brackets may be used to define a set of characters.

Within a set definition, the '-' symbol may be used to denote a character range. For example, [0123456789] is the set of all decimal digits, and [a-zA-Z] is the set of all upper and lower-case letters.

(c) **Or.**

This operation allows us to specify alternatives in the pattern.

If the '|' symbol appears between two characters, then there is a match if and only if either of the characters appears in the text.

(d) **Closure.**

This operation allows parts of a pattern to be repeated arbitrarily. If we have the closure symbol '*', then there is a match if and only if the symbol occurs zero or more times. If we have the closure symbol '+', then there is a match if the symbol occurs one or more times. Closure will be denoted by placing the * or + symbol after the character, set, or group to be repeated.

2.2 7064 Grammar

The 7064 command language is formally described by a "grammar", i.e. a set of "production rules" that concisely describes the set of strings that are valid 7064 command sequences.

The rules consists of: "terminal" symbols like '(', ';', 7064 command names, and digits; "non-terminal" symbols like <cmd>, <unsigned integer const>, and <trig_cmd>, which are internal to the grammar, are "metasymbols", which describe the meaning of the productions.

Within production rules, character literals are enclosed in single quotes '' in order to distinguish them from metasymbols.

One non-terminal symbol (in this case <cmdlist>) is "distinguished" in the sense that a string of terminal symbols is a valid 7064 command if there is some way to use the productions to "derive" that string from the distinguished non-terminal symbol (i.e. <cmdlist>) by replacing (in any number of steps) a non-terminal by any of the "or" clauses on the right-hand side of a production for that non-terminal symbol.

2.3 Metasymbols

=> "is a", defines the left-hand side of a rule in terms of the right-hand side.

{} Grouping

| Alternation

* Closure, zero or more times

+ Closure, one or more times

2.4 Tokens

A token is a textual unit. There are four classes of tokens: identifiers, commands, constants, and other separators. Blanks, horizontal and vertical tabs, new lines, formfeeds, and comments as described below (collectively, "white space") are ignored except as they serve to separate tokens. Some whitespace is required to separate otherwise adjacent identifiers, keywords, and constants.

2.5 Comments

The 7064 command parser accepts C-style comments, /* */. Comments may span across multiple lines; they do not nest.

3.0 Sample Command Strings

These are some examples of command strings that can be sent to the 7064 via the VXIbus word serial protocol. Note that the 7064 software recognizes and ignores C-style comments.

```
rstbit (00000, 0001);   /* Placement of whitespace is not critical */
setbit( 0002,001) ;

crstbit(#b00001010, 1, 2);    /* Illustrates different numerical formats */
csetbit(#Q7, 0x0c, 1) ;

outbit(8,1,0);          /* All identifiers are case-insensitive, so these two */
OUTBIT(8,1,0);          /* commands are identical */
```

```
inport(11);          /* After executing any 7064 input command, the 7064 makes */ /* the data
                        read from the port available to be read by the controller */
cinbit(#b0111, 8, 0);

outbit(2, 3, 1);     /* More examples of numerical formats */
outportb(11, #q377);
outport(11,  0377);
coutport(0, 11, 0x7f);

PULSEHI(4, 5);
PULSELO(6, 7);
cpulsehi(0, 4, 5);
cpulselo(0, 6, 6);

DOWNLOAD myfunc();
S00600004844521B
S2130004002E7C0040FFFE46FC27004BF900400014
S804000400F7
myfunc();   /* Calls "myfunc", which was loaded at address 0x00000400 */
            /* Note:  this would probably not produce the desired result because */
            /* 0x400 is ROM */

triggen(); triggin(1); trigout(7); /* Commands may appear on the same line */

base(16); /* Subsequent 7064 numerical output will be in hex */
      /* After executing memory "peek" commands, the 7064 makes the data */
      /* read from the specified memory locations available to the controller.*/

peek(0xe000);
peekb( 0xf0f0 );
poke(0xe000, #H1F);
pokeb( 0xf0f0, #b00011111 );

/* Switch to the user parse mode */
USR_SCAN();
some user cmds can go here ...
VT1_SCAN();

/* We're now back in the 7064 command mode */
myfunc();
trigout(3);
```

APPENDIX C

ERROR HANDLING

Errors can be classified according to type and severity; they are denoted by unsigned integer values referred to as 'error descriptors'. See the error.h file for a complete description.

The VT1 associates a message string with each error descriptor. This message string is a 'printf-style' format string, i.e. it can contain embedded "%" format specifications. At run-time, additional arguments passed to the error handler are used as 'printf' parameters, thus providing supplemental information in the error message. In the simplest case the message is a simple string that doesn't contain any embedded "%" format specifiers, and thus no additional arguments are required.

execerror() is the executive error management function. It's called to process all classes of VT1 errors. It simply transmits its arguments to, and invokes, an error handler. It is the error handler's responsibility to carry out the actual error processing. By default, the normal VT1 error handler is invoked. The user may select an application-specific error handler with the set_error_handler() function, however. In this case the application-specific error handler is invoked with the same arguments as the default VT1 error handler, and is free to process the errors in any way it sees fit.

Default VT1 error handling:

The default VT1 error handler performs the following operations:

a. Prefixes a short string onto the message denoting the error type. For example, "Warning: ", and "Syntax error: ".

b. Expands additional 'printf-style' error message arguments, if any.

c. Prints the fully-constructed error message to the output buffer.

d. If the E_LINEDUMP bit is set in the error descriptor, then the current input line is printed too. This can assist the user in locating the actual source of the error.

e. Calls signal_err() to alert VXI controller of an error condition.
f. If the error is fatal, it "locks up" in an infinite loop. All interrupts remain enabled, however.

Interface to error handler:

The error handler is passed the error descriptor and a pointer to a variable argument list containing additional 'printf' paramaters, if any. The application-specific error handler can use the macros provided in error.h to analyze the error descriptor and take appropriate action; it's also free to ignore additional 'printf' arguments, if any exist. It is called with this interface:

```
application_error_handler(unsigned errdes, void *argp)

    where,
      errdes : the error descriptor
      argp   : a pointer to a printf-style variable argument list
```

The Intermetrics C cross compiler supports neither the ANSI-standard nor the original UNIX-style "variable argument" lists. A proprietary scheme is used instead; the declarations and macros are found in the file "rt.h". They are:

ARGPTYPE: Data type of a variable-argument-list pointer ARG() : Fetches an argument from a variable argument list ARGINC(): Increments pointer to point to next argument in list

PUBLIC FUNCTIONS:
execerror() : Executive error handler
set_error_handler() : Set application-specific error handler

PRIVATE FUNCTIONS:

Strictly speaking, these functions should be declared 'static'. However, they have been declared global in case the application-specific error handler wishes to make use of them.

```
error_handler()     : Default VT1 error handler
error_msg_lookup()  : Lookup error message string given error descriptor
```

SEE ALSO:
error.h: VXI Tech. error include file.
rt.h : Intermetrics "run-time", contains non-standard 'var-args' package

REFERENCES:

"The C Programming Language", 2nd ed., by K&R.
p 155: Example usage of variable argument lists
p 202: Argument promotion rules for variable argument lists
p 254: Specification of ANSI <stdarg.h> package

DESCRIPTION: Package interface to run-time error module

An error descriptor is an unsigned integer composed of 4 bit fields:

field 1) Error severity field, 1 bit
field 2) Error type field, 2 bits
field 3) Line dump field, 1 bit
field 4) Error number, all remaining bits (2^12 for 16-bit unsigned ints)

The value of '0' is a reserved descriptor that denotes 'SUCCESS', i.e. no error condition.

In defining bit fields and macros associated with error descriptors, care must be taken to ensure portability between host an target environments, since the size of an integer may not be the same.

NOTE! For backwards compatibility with old versions of VTX firmware:
1) Never delete an existing error descriptor.
2) Never redefine an existing error descriptor.
3) Always add new error descriptors to the end of the list.
4) Never define a 0-valued descriptor. 0 is a reserved descriptor.

PUBLIC MACROS:
errortype(ed) : returns error type
isfatal(ed) : TRUE if error descriptor denotes a fatal error
iswarning(ed) : TRUE if error descriptor denotes a warning
issyntax(ed) : TRUE if error descriptor denotes a syntax error
isruntime(ed) : TRUE if error descriptor denotes a run-time error
isinternal(ed) : TRUE if error descriptor denotes an internal error
issuccess(ed) : TRUE if error descriptor denotes successful operation
iserror(ed) : TRUE if error descriptor denotes any kind of error
dolinedump(ed) : if TRUE, dump current input line along/w error message
---------- */

```
#ifndef ERROR_INCLUDE
#define ERROR_INCLUDE

#define E_NBITS    (sizeof(unsigned)*8)

/*
  Error severity:
     Error severity is considered either fatal or non-fatal. This
     information is encoded into a 1-bit field.
```

```
*/
#define E_SEVERITY_FSIZE        1u /* field size */
#define E_SEVERITY_BITMASK      (1u << (E_NBITS-E_SEVERITY_FSIZE))
#define E_FATAL                 (1u << (E_NBITS-E_SEVERITY_FSIZE))

/*
  Error types: The error type is encoded into a 2-bit field.
*/
#define E_TYPE_FSIZE        2u      /* field size */
#define E_TYPE_BITMASK  /* 3 = 2^2 - 1: all bits in the field set to 1s */ \
        (3u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE)))
#define E_WAR (0u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE))) /* warning  */
#define E_SYN (1u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE))) /* syntax   */
#define E_RUN (2u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE))) /* run time */
#define E_INT (3u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE))) /* internal */

/*
   Line dump: if set to '1', then the current input line is dumped along with
        the error message. This can be useful to help pinpoint the location
        of the error.
*/
#define E_LINEDUMP_FSIZE    1u      /* field size */
#define E_LINEDUMP_BITMASK  \
        (1u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE+E_LINEDUMP_FSIZE)))
#define E_LINEDUMP \
        (1u << (E_NBITS-(E_TYPE_FSIZE+E_SEVERITY_FSIZE+E_LINEDUMP_FSIZE)))
/*
  Error macros:
     The macro argument to each of these is an error descriptor.
*/
#define isfatal(ed)       ((ed) & E_SEVERITY_BITMASK)
#define errortype(ed)     ((ed) & E_TYPE_BITMASK)
#define iswarning(ed)     (errortype((ed)) == E_WAR)
#define issyntax(ed)      (errortype((ed)) == E_SYN)
#define isruntime(ed)     (errortype((ed)) == E_RUN)
#define isinternal(ed)    (errortype((ed)) == E_INT)
#define dolinedump(ed)    ((ed) & E_LINEDUMP_BITMASK)
#define SUCCESS           0u
#define issuccess(ed)     ((ed) == SUCCESS)
#define iserror(ed)       (!issuccess((ed)))
```

```
/*
   VTX error descriptors:
      A block of 200 error numbers has been arbitrarily allocated to each
      type of error.

   NOTE! For backwards compatibility with old versions of VTX firmware:
      1) Never delete an existing error descriptor.
      2) Never redefine an existing error descriptor.
      3) Always add new error descriptors to the end of the list.
      4) Never define a 0-valued descriptor. 0 is a reserved descriptor.
*/

/*
  Warnings: [1 .. 199] Warnings are caused for a variety of reasons. The
      VTX is able to continue execution.
*/
#define E_NAME                  ( 1u | E_WAR | E_LINEDUMP)
/*      next warning number is ...   2                  */

/*
  Syntax errors: [200 .. 399] Syntax errors are discovered by the parser or
      scanner, and the VTX is able to continue execution.
*/
#define E_SYNTAX                (200u | E_SYN | E_LINEDUMP)
#define E_UNDEFINED_FUNC        (201u | E_SYN | E_LINEDUMP)
#define E_UNDEFINED_VARIABLE    (202u | E_SYN | E_LINEDUMP)
#define E_NUMERICAL_FORMAT      (203u | E_SYN | E_LINEDUMP)
#define E_NARGS                 (204u | E_SYN | E_LINEDUMP)
#define E_EXPECTED              (205u | E_SYN | E_LINEDUMP)
#define E_REDEFINED             (206u | E_SYN | E_LINEDUMP)
/*      next syntax error is ...   207                  */

/*
  Run-time errors: [400 .. 599] Run-time errors can be caused by out-of-range
      arguments passed to built-in functions and other abnormal and exception
      run-time conditions. The VTX may or may not be able to continue
      execution.
```

```
*/
#define E_TRIG_LINE              (400u | E_RUN | E_LINEDUMP)
#define E_BASE_VALUE             (401u | E_RUN | E_LINEDUMP)
#define E_VECTORNUM              (402u | E_RUN | E_LINEDUMP)
#define E_VXIREG                 (403u | E_RUN | E_LINEDUMP)
#define E_GLOBAL_SIZE            (404u | E_RUN | E_LINEDUMP)
#define E_HEAP_SIZE              (405u | E_RUN | E_LINEDUMP)
#define E_INQ_SIZE               (406u | E_RUN | E_LINEDUMP)
#define E_NVCODE_SIZE            (407u | E_RUN | E_LINEDUMP)
#define E_OUTQ_SIZE              (408u | E_RUN | E_LINEDUMP)
#define E_SECQ_SIZE              (409u | E_RUN | E_LINEDUMP)
#define E_CARDNUM                (410u | E_RUN | E_LINEDUMP)
#define E_CARDPORTNUM            (411u | E_RUN | E_LINEDUMP)
#define E_PORTNUM                (412u | E_RUN | E_LINEDUMP)
#define E_OUTOF_MEMORY           (413u | E_RUN | E_FATAL)
#define E_ARITH_STACK_OVERFLOW   (414u | E_RUN | E_FATAL)
#define E_CODESEG_OVERFLOW       (415u | E_RUN | E_FATAL)
#define E_NVFLAG                 (416u | E_RUN | E_LINEDUMP)
/*      next run-time error is ... 417                  */


/*
  Internal errors: [600 .. 799]. Internal errors are caused by unanticipated
       faults in the VTX firmware. They should never occur; they are always
       fatal.
*/
#define E_ARITH_STACK_UNDERFLOW  (600u | E_INT | E_FATAL)
/*      next internal error is     601              */

/* ------------------------------------------------------------------------ */

/*
   Size of the error message buffer. NOTE! The length of a printf-style
   error message string should not exceed 1/2 this size!
*/
#define ERRBUF_SIZE    256

#ifdef ERROR_TABLE
static struct errstruct {
   unsigned errdes;          /* error descriptor              */
   char    *errmsg;          /* & its associated error message */
} Error_table[] = {
/* Warnings */
{E_NAME, "name too long\n" },
```

```
/* Syntax errors */
{E_SYNTAX, "\n" /* generic syntax error */ },
{E_UNDEFINED_FUNC, "\"%s()\", undefined function\n" },
{E_UNDEFINED_VARIABLE, "\"%s\", undefined variable\n" },
{E_NUMERICAL_FORMAT, "illegal numerical format\n" },
{E_NARGS, "function %s() accepts %d arguments\n" },
{E_EXPECTED, "expected `%s'\n" },
{E_REDEFINED, "%s already defined\n" },

/* Run-time errors */
{E_TRIG_LINE, "trigger line out of range, expression evaluates to %u\n" },
{E_BASE_VALUE, "base value out of range, expression evaluates to %u\n" },
{E_VECTORNUM, "vector number out of range, expression evaluates to %u\n" },
{E_VXIREG, "VXI register number out of range, expression evaluates to %u\n" },
{E_GLOBAL_SIZE, "global size out of range, expression evaluates to %u\n" },
{E_HEAP_SIZE, "heap size out of range, expression evaluates to %u\n" },
{E_INQ_SIZE, "input queue size out of range, expression evaluates to %u\n" },
{E_NVCODE_SIZE, "non-vol code size out of range, expression evaluates to %u\n"},
{E_OUTQ_SIZE, "output queue size out of range, expression evaluates to %u\n" },
{E_SECQ_SIZE,"secondary queue size out of range, expression evaluates to %u\n"},
{E_CARDNUM, "card number out of range, expression evaluates to %u\n" },
{E_CARDPORTNUM, "card-relative port number out of range, expression evaluates to %u\n"
},
{E_PORTNUM, "port number out of range, expression evaluates to %u\n" },
{E_OUTOF_MEMORY, "out of memory in file %s, line %d\n" },
{E_ARITH_STACK_OVERFLOW, "expression too complex, arithmetic stack overflow\n"},
{E_CODESEG_OVERFLOW, "interpreter program too large, code segment overflow\n"},
{E_NVFLAG, "NONVOL flag number out of range, expression evaluates to %u\n" },

/* Internal errors */
{E_ARITH_STACK_UNDERFLOW, "arithmetic stack underflow\n" },
};
#define MAX_ERRORS      (sizeof(Error_table)/sizeof(struct errstruct))
#endif  /* error table */

extern void execerror(unsigned errdes, ...);
#ifdef HOST_ENV
#   include <stdarg.h>
    extern void error_handler(unsigned errdes, va_list argp);
    extern void set_error_handler(void (*errfunc)(unsigned, va_list));
#else /* Intermetrics doesn't support ANSI standard var-args package */
    extern void error_handler(unsigned errdes, void *argp);
    extern void set_error_handler(void (*errfunc)(unsigned, void*));
#endif
```

```
extern char *error_msg_lookup(unsigned errdes);

#endif  /* ERROR_INCLUDE */
```

Racal Instruments

REPAIR AND CALIBRATION REQUEST FORM

To allow us to better understand your repair requests, we suggest you use the following outline when calling and include a copy with your instrument to be sent to the Racal Repair Facility.

Model_____ Serial No._____ Date_____

Company Name_____ Purchase Order #_____

Billing Address_____

City

State/Province Zip/Postal Code Country

Shipping Address_____

City

State/Province Zip/Postal Code Country

Technical Contact_____ Phone Number ( )_____

Purchasing Contact_____ Phone Number ( )_____

1. Describe, in detail, the problem and symptoms you are having. Please include all set up details, such as input/output levels, frequencies, waveform details, etc.

2. If problem is occurring when unit is in remote, please list the program strings used and the controller type.

3. Please give any additional information you feel would be beneficial in facilitating a faster repair time (i.e., modifications, etc)

4. Is calibration data required? Yes No (please circle one)

Call before shipping

Note: We do not accept "collect" shipments.

Ship instruments to nearest support office listed on back.

Support Offices

Racal Instruments, Inc.
4 Goodyear Street
Irvine, **CA** 92618-2002
Phone: 714-859-8999
800 RACAL-ATE
800-722-2528
FAX: 714-859-7139

Racal Instruments Ltd
480 Bath Road
Slough, Berkshire
SL1 6BE, **England**
Phone: +44 (0) 1628 604455
FAX: +44 (0) 1628 662017

Racal Systems Electronique s.a.
18 Avenue Dutartre
78150 LeChesnay, **France**
Phone: +33 (1) 3955 8888
FAX: +33 (1) 3955 6735

Racal Systems Elettronica Srl
Strada 2-Palazzo C4
20090 Milanofiori Assago
Milan, **Italy**
Phone: +39 (2) 5750 1796
FAX: +39 (2) 5750 1828

Racal Elektronik System GmbH
Frankenforster Strasse 21
51427 Bergisch-Gladbach 1,**Germany**
Phone: +49 2204 92220
FAX: +49 2204 21491

Racal Australia Pty Ltd
3 Powells Road
Brookvale, NSW 2100, **Australia**
Phone: +61 (2) 9936 7000
FAX: +61 (2) 9936 7036

Racal Electronics Pte Ltd
26 Ayer Rajah Crescent, 04-06/07
Ayer Rajah Industrial Estate, **Singapore** 0513
Phone: +65 7792200
FAX: +65 7785400

Racal Instruments, Ltd.
Sun House 13th Floor
181 Des Voeux Road, Central, **Hong Kong**
Phone: +852 281 58663
FAX +852 281 58141

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP