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 *Intel iSBC CSM/001* at our website: ***Click HERE***

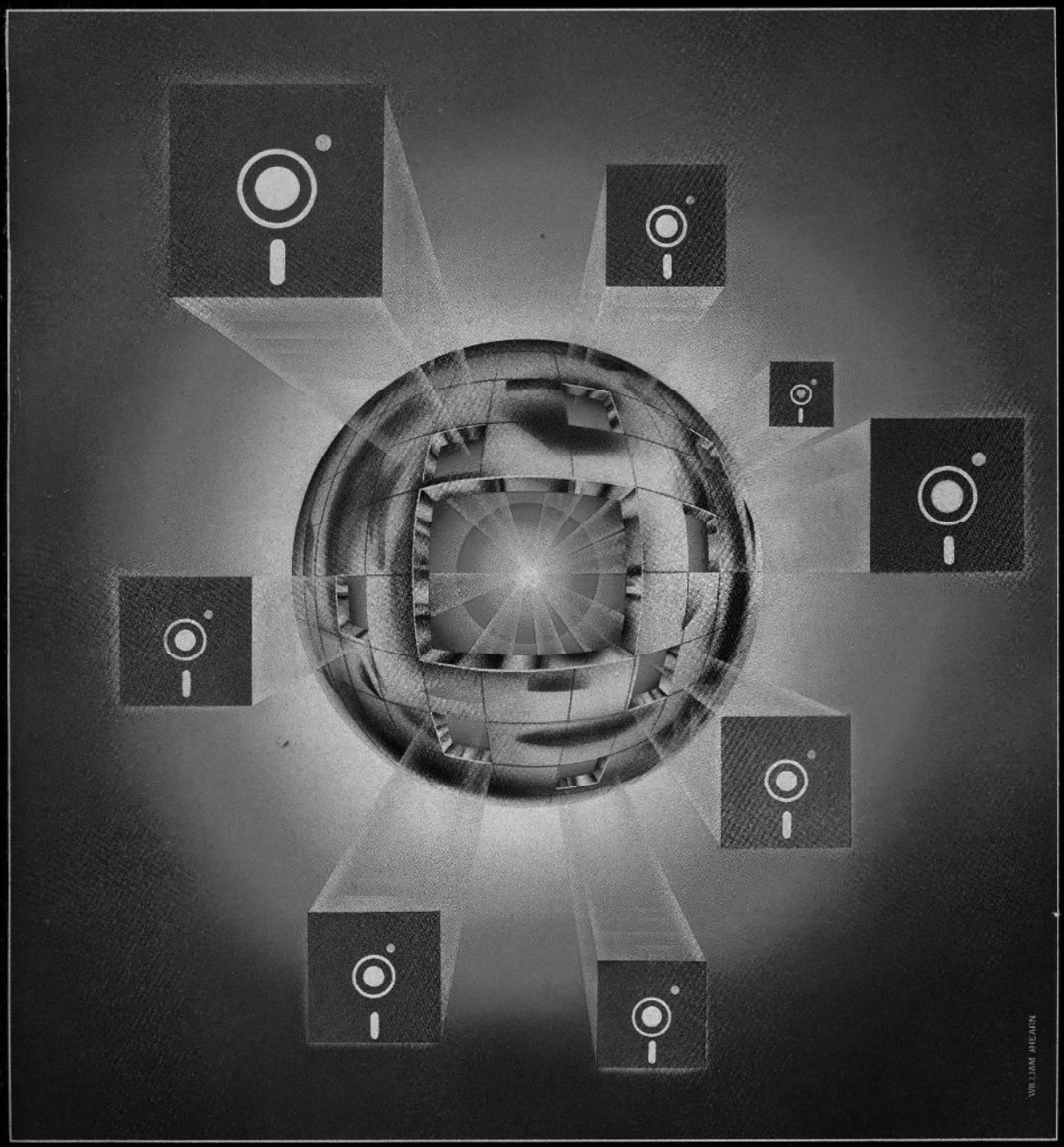
intel

Software Handbook

1985

intel

Software Handbook



WILLIAM AHEARN

Order Number: 230786-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LITERATURE

In addition to the product line handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846-003) provides an overview of Intel's complete product lines and customer services.

Consult the INTEL LITERATURE GUIDE (Order No. 210620) for a listing of Intel literature. TO ORDER literature in the U.S., write or call the INTEL LITERATURE DEPARTMENT, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

Use the order blank on the facing page or call our TOLL FREE number listed above to order literature. Remember to add your local sales tax.

1985 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

	*U.S. PRICE
QUALITY/RELIABILITY HANDBOOK (Order No. 210997-001) Contains technical details of both quality and reliability programs and principles.	**$15.00**
CHMOS HANDBOOK (Order No. 290005-001) Contains data sheets only on all microprocessor, peripheral, microcontroller and memory CHMOS components.	**$12.00**
MEMORY COMPONENTS HANDBOOK (Order No. 210830-004)	**$18.00**
TELECOMMUNICATION PRODUCTS HANDBOOK (Order No. 230730-003)	**$12.00**
MICROCONTROLLER HANDBOOK (Order No. 210918-003)	**$18.00**
MICROSYSTEM COMPONENTS HANDBOOK (Order No. 230843-002) Microprocessors and peripherals—2 Volume Set	**$25.00**
DEVELOPMENT SYSTEMS HANDBOOK (Order No. 210940-003)	**$15.00**
OEM SYSTEMS HANDBOOK (Order No. 210941-003)	**$18.00**
SOFTWARE HANDBOOK (Order No. 230786-002)	**$12.00**
MILITARY HANDBOOK (Order No. 210461-003) Not available until June.	**$15.00**
COMPLETE SET OF HANDBOOKS (Order No. 231003-002) Get a 25% discount off the retail price of $160.	**$120.00**

***U.S. Price Only**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel®

U.S. LITERATURE ORDER FORM

NAME: _____ TITLE: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: (____) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL
☐☐☐☐☐☐-☐☐☐				
☐☐☐☐☐☐-☐☐☐				
☐☐☐☐☐☐-☐☐☐				
☐☐☐☐☐☐-☐☐☐				
☐☐☐☐☐☐-☐☐☐				
☐☐☐☐☐☐-☐☐☐				

POSTAGE AND HANDLING:
Add appropriate postage and handling to subtotal
10% U.S.
20% Canada

Subtotal _____

Your Local Sales Tax _____

_____

Allow 4-6 weeks for delivery

Total _____

Pay by Visa, MasterCard, Check or Money Order, payable to Intel Literature. Purchase Orders have a $50.00 minimum.

☐ Visa ☐ MasterCard Account No. _____ Expiration Date _____

Signature: _____

Mail To: Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of this book.

For information on quantity discounts, call the 800 number below:
TOLL-FREE NUMBER: (800) 548-4725
Prices good until 12/31/85.
Source HB

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Mail To: Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



SOFTWARE HANDBOOK

1985

About Our Cover:
The design on our front cover is an abstract portrayal of the role Intel software plays in systems development. The heart of systems development is surrounded by a sphere of peripheral development through which Intel software can guide the designer to reaching development goals.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a) (9). Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, i, ICE, iCS, iDBP, iDIS, I²ICE, iLBX, i_m, iMDDX, iMMX, Insite, Intel, int_el, int_elBOS, Intelevision, int_elligent Identifier, int_eligent Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MCS, Megachassis, MICRO-MAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, OpeNet, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or UPI and a numerical suffix.

The following are trademarks of the companies indicated and may only be used to identify products of the owners.

CP/M is a trademark of Digital Research, Inc.

DEC, DEC-10, DEC-20, PDP-11, DECnet, DECwriter, RSTS, and VAX are trademarks of Digital Equipment Corporation.

MDS is an ordering code only and is not used as a product name or trademark.

MDS® is a registered trademark of Mohawk Data Sciences Corporation.

Microsoft is a trademark of Microsoft, Inc.

©Intel Corporation, 1984

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel®

Table of Contents

CHAPTER 1
OVERVIEW
Introduction 1-1

CHAPTER 2
OPERATING SYSTEMS
Introduction 2-1
8080/8085 Microprocessor Family
DATA SHEET
Digital Research Inc. CP/M 2.2 Operating System 2-2
8086/8088/80286 Microprocessor Family
DATA SHEETS
iRMX 86 Operating System 2-5
iOSP 86, iAPX 86/30, iAPX 88/30, iAPX 186/30 and iAPX 188/30 Support Package 2-24
iRMX 86 MULTIBUS II Support Package 2-28
FACT SHEET
iRMX Operating Systems 2-31
XENIX 3.0 Operating Systems 2-37
APPLICATION NOTES
AP-130 Using Operating Systems Processor's to Simplify Microcomputer Designs 2-44
AP-174 Optimizing the iRMX 86 Operating System Performance on System 86/310 and System 86-330 2-95
AP-184 Writing Device Drivers for XENIX 86 and 286 - Task or Trivia? 2-123
AP-221 An Introduction to Task Management in the iRMX 86 Operating System 2-193
ARTICLE REPRINTS
AR-286 Software That Resides in Silicon 2-234
AR-287 Putting Real-Time Operating Systems to Work 2-240
AR-288 Intel's Matchmaking Strategy: Marry iRMX Operating System with Hardware 2-252
AR-337 Industrial PC Systems Demand Real Time Operating Systems 2-255

CHAPTER 3
TRANSLATORS AND UTILITIES FOR PROGRAM DEVELOPMENT
Introduction 3-1
MCS-80/85 Microprocessor Family
DATA SHEETS
PL/M 80 High Level Programming Language 3-3
FORTRAN 80 8080/8085 ANS FORTRAN 77 Intellec Resident Compiler 3-6
Microsoft, Inc. BASIC-80 Interpreter Software Package 3-10
Microsoft, Inc. BASIC-80 Compiler Software Package 3-13
Microsoft Multiplan Spreadsheet 3-15
PASCAL-80 Software Package 3-23
WORDSTAR Word Processing Software 3-28
iAPX 86/88/186/188/286 Microprocessor Family
DATA SHEETS
iAPX 86, 88 Software Development Packages for Series II/PDS 3-35
86/88/186/188 Software Packages 3-45
FORTRAN 86/88 Software Package 3-46
Pascal 86/88 Software Package 3-50
PL/M 86/88/186/188 Software Package 3-53
iC-86 C Compiler for the 8086 3-57
8087 Support Library 3-61
8087 Software Support Package 3-65
8089 IOP Software Support Package #407200 3-68
iAPX 286 Software Development Package 3-71
PL/M 286 Software Package 3-76
iSDM 286 iAPX 286 System Debug Monitor 3-80
80287 Support Library 3-84
Pascal-286 Software Package 3-87
VAX/VMS Resident Software Development Packages for iAPX 286 3-90

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VAX/VMS Resident iAPX 86/88/186 Software Development Packages 3-96
iSDM 86 System Debug Monitor 3-103
iVDI 720 Graphics Virtual Device Interpreter 3-108
iPLP 720 NAPLPS Interpreter 3-112
FACT SHEETS
iRMX Languages 3-116
XENIX Languages 3-121
Single Chip Microcontroller Software
DATA SHEETS
2920 Software Support Package 3-125
MCS-48 Diskette-Based Software Support Package 3-136
8051 Software Package 3-138
iRMX 51 Real-Time Multitasking Executive 3-147
MCS-96 Software Development Packages 3-153

CHAPTER 4
DEVELOPMENT PRODUCTIVITY TOOLS
Introduction 4-1
Program Development and Management Tools
DATA SHEETS
PSCOPE High-Level Program Debugger 4-2
Program Management Tools 4-7
ISIS-II Software Toolbox 4-10
8086 Software Toolbox 4-12
AEDIT Text Editor 4-14

CHAPTER 5
COMMUNICATION SOFTWARE
Introduction 5-1
DATA SHEETS
Mainframe Link for Distributed Development 5-2
Intel Asynchronous Communications Link 5-5
iNA 960 Network Software 5-8
NDS II Electronic Mail 5-20
iNA 955 iRMX NDS-II Link 5-22
iRMX 510 iDCM Support Package 5-26

CHAPTER 6
SYSTEM AND APPLICATIONS SOFTWARE
Introduction 6-1
FACT SHEETS
XENIX Productivity Software Tools 6-2
Third Party Software for Intel Products 6-10
Database Information System iDIS 715 6-13

CHAPTER 7
COMPONENT SOFTWARE
DATA SHEETS
80130/80130-2 iAPX 86/30, 88/30, 186/30, 188/30 iRMX 86
Operating System Processors 7-1
80150/80150-2 iAPX 86/50, iAPX 88/50, 186/50, 188/50 CP/M 86
Operating System Processors 7-23

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 8
USER LIBRARY
Introduction 8-1
User Library
Insite User's Program Library 8-2
Insite Submittal Requirements 8-3
Insite Index of Program 8-5

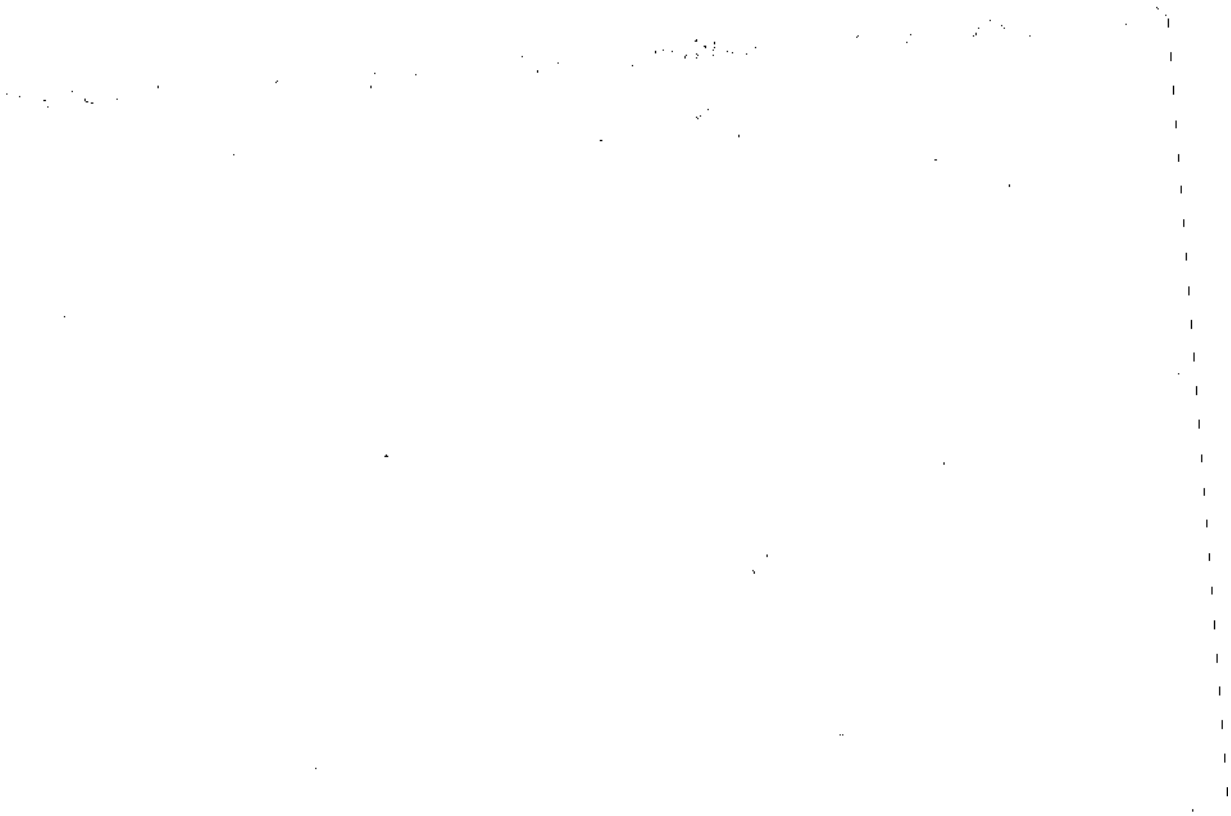
APPENDIX A
Software Standards A-1
Software Support Services A-3
iRUG A-5

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Overview 1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel®

SOFTWARE HANDBOOK OVERVIEW

Welcome to the Intel Software Handbook. This handbook is a complete guide to the software products and services offered by Intel.

Intel's software products follow the open systems strategy that allows Intel products to be purchased at the customers' desired level of integration. Hence these products are available for component, board, or systems applications. This open systems philosophy is backed by software standards that insure that the software can operate at numerous levels of integration. These software standards are described in the appendix.

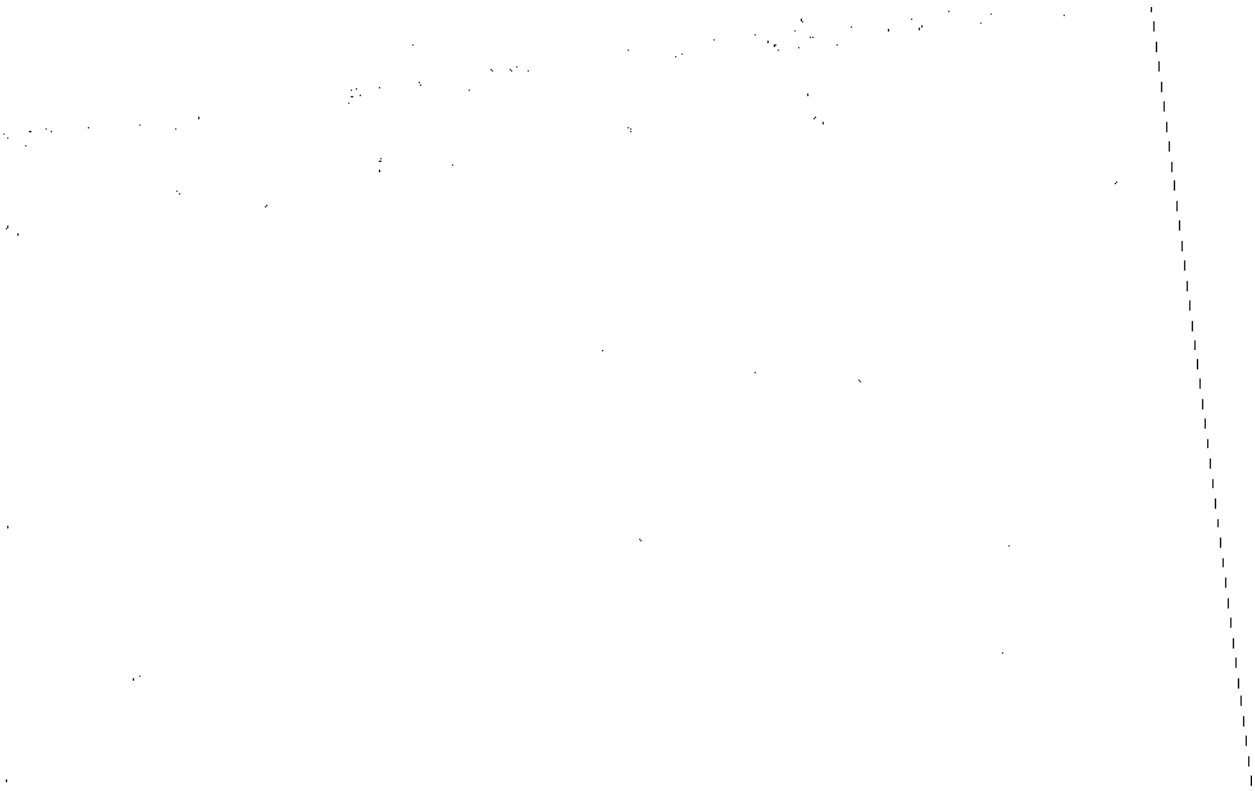
Software for Intel's products is available both from Intel and from Independent Software Vendors (ISVs). For a complete listing of software available from ISVs, see the Intel Yellow Pages which is published annually by Intel. This handbook describes software products that are available through Intel, consisting of Intel-developed and ISV-developed products. Products that are offered by Intel have all been evaluated and tested to meet Intel's quality standard. They are backed by an extensive support organization described in the appendix.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Operating Systems 2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OPERATING SYSTEMS

INTRODUCTION

The ability to convert advanced microprocessor technology into solutions for modern day problems begins with effective and efficient designs for new hardware products and architecture. However, a most critical element in the success of any microcomputer solution is the availability of a high quality, reliable operating system. Without this software counterpart, the technological advances cannot be fully implemented, nor their benefits fully realized.

The classic role of the microcomputer operating system can be outlined by viewing its major functions and purposes. The functions of the microcomputer operating system are threefold: 1) to manage system resources and the allocation of these resources to users; 2) to provide automatic functions such as initialization and start-up procedures; and 3) to provide an efficient, straightforward and consistent method for user programs to interface with the hardware subsystems, including a simple and friendly human interface. Typically, the operating systems have one of two main purposes. First, they can be used to develop a new software system that runs on another machine. These systems are usually large and fairly sophisticated. ISIS and *XENIX are examples of such developmental operating systems. The second purpose for microcomputer operating systems is directed toward the execution of software programs for targeted application. The largest number of operating systems are of this type, including the RMX systems. The most critical requirement is for these systems to be effective and efficient since they are usually small, fast systems dedicated to a specific real-time application.

This rather neat and simple categorization of microcomputer operating systems, which has been useful in the past, is quickly becoming blurred. The rapid developments in microcomputer technology have increased the power and decreased the cost of microcomputers, allowing them to become applicable to the solution of a broader variety and more sophisticated set of problems. Microcomputer systems must increasingly provide such capabilities as multiprogramming, multitasking, multiprocessing, networking, as well as scheduling and priority determination. As systems become more complex, they must still remain responsive to real-time applications. Operating systems must be able to capitalize on the trends toward placing more and more software into silicon. This trend is blurring the distinction between the hardware and software subsystems. Microcomputer systems are also evolving to encompass both the developmental and target application purposes into one system.

These dramatic changes in technology place additional demands on operating systems. We see operating systems undergoing changes to consider the need for: 1) modularity and ease of configurability; 2) evolutionary, not revolutionary, path of growth; and 3) standardization in languages, networks and the operating system itself. The first need is required to allow the system to be a powerful development tool yet configurable to more specialized applications. The last two items are needed to provide protection of a firm's software investment, including the option to move toward silicon software.

The operating systems and executives in this section are state-of-the-art microcomputer systems that have taken to task the challenges posed by advancing microprocessor technology. These operating systems offer the widest range of solutions with the highest quality and most future-oriented software available today. Consequently, our customers can select the appropriately optimized option to achieve their price/performance goals and give them time-to-market advantage over their competitors.

*XENIX is a trademark of Microsoft Corp.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DIGITAL RESEARCH INC.
CP/M* 2.2 OPERATING SYSTEM

- **High-performance, single-console operating system**
- **Simple, reliable file system matched to microcomputer resources**
- **Table-driven architecture allows field reconfiguration to match a wide variety of disk capacities and needs**
- **Extensive documentation covers all facts of CP/M applications**
- **More than 1,000 commercially available compatible software products**
- **General-purpose subroutines and table-driven data-access algorithms provide a truly universal data management system**
- **Upward compatibility from all previous versions**

CP/M 2.2 is a monitor control program for microcomputer system and application uses on Intel 8080/8085-based microcomputer. CP/M provides a general environment for program execution, construction, storage, and editing, along with the program assembly and check-out facilities.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this system, a large number of distinct programs can be stored in both source- and machine-executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Nearly all personal software programs can be bought configured to run under CP/M, several of which are available from Intel.

FEATURES

CP/M is logically divided into four distinct modules:

BIOS—Basic I/O System

—Provides primitive operations for access to disk drives and interface to standard peripherals (teletype, CRT, paper tape reader/punch, bubble memory, and user-defined peripherals)

—Allows user modification for tailoring to a particular hardware environment

BDOS—Basic Disk Operating System

—Provides disk management for one to sixteen disk drives containing independent file directories

—Implements disk allocation strategies for fully dynamic file construction and minimization of head movement across the disk

—Uses less than 4K of memory allowing plenty of memory space for applications programs

—Uses less than 4K of memory

—Makes programs transportable from system to system

—Entry points include the following primitive operations which can be programmatically accessed:

SEARCH	Look for a particular disk file by name
OPEN	Open a file for further operations
CLOSE	Close a file after processing
RENAME	Change the name of a particular file
READ	Read a record from a particular file
WRITE	Write a record to a particular file
SELECT	Select a particular disk drive for further operations

© INTEL CORPORATION 1983

SEPTEMBER 1984
ORDER NUMBER:210268-004

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CCP—Console Command Processor

—Provides primary user interface by reading and interpreting commands entered through the console

—Loads and transfers control to transient programs, such as assemblers, editors, and debuggers

—Processes built-in standard commands including:

ERA	Erase specified files
DIR	List file names in the directory
REN	Rename the specified file
SAVE	Save memory contents in a file
TYPE	Display the contents of a file on the console

TPA—Transient Program Area

—Holds programs which are loaded from the disk under command of the CCP

—Programs created under CP/M can be checked out by loading and executing these programs in the TPA

—User programs, loaded into the TPA, may use the CCP area for the program's data area

—Transient commands are specified in the same manner as built-in commands

—Additional commands can be easily defined by the user

—Defined transient commands include:

PIP	Peripheral Interchange Program —implements the basic media transfer operations necessary to load, print, punch, copy, and combine disk files. PIP also performs various reformatting and concatenation functions. Formatting options include parity-bit removal, case conversion, Intel hex file validation, subfile extraction, tab expansion, line number generation, and pagination
ED	Text Editor—allows creation and modification of ASCII files using extensive context editing commands: string substitution, string search, insert, delete and block move; ED allows text to be located by context, line number, or relative position with a macro command for making extensive text changes with a single command line
ASM	Fast 8080 Assembler—uses standard Intel mnemonics and pseudo operations with free-format input, and conditional assembly features
DDT	Dynamic Debugging Tool—contains an integral assembler/disassembler module that lets the user patch and display memory in either assembler mnemonic or hexadecimal form and trace program execution with full register and status display; instructions can be executed between breakpoints in real-time, or run fully monitored, one instruction at a time
SUBMIT	Allows a group of CP/M commands to be batched together and submitted to the operating system by a single command
STAT	Lists the number of bytes of storage remaining on the currently logged disks, provides statistical information about particular files, and displays or alters device assignments
LOAD	Converts Intel hex format to absolute binary, ready for direct load and execution in the CP/M environment
SYSGEN	Creates new CP/M system disks for back-up purposes
MOVCPM	Provides regeneration of CP/M systems for various memory configurations and works in conjunction with SYSGEN to provide additional copies of CP/M

BENEFITS

—Easy implementation on any computer configuration which uses an Intel 8080/8085 Central Processing Unit (see the CP/M-86 data sheet for CP/M applications on the iAPX86 CPU)

—iPDS version supports bubble memory option as an additional diskette drive. Also allows diskette duplication with a single drive

—Extensive selection of CP/M-compatible programs allows production and support of a comprehensive software package at low cost

—Field programmability for special-purpose operating system requirements

—Upward compatibility from previous versions of CP/M release 1

AFN-02111C

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

—Provides field specification of one to sixteen logical drives, each containing up to eight megabytes

—Files may contain up to 65,536 records of 128 bytes each but may not exceed the size of any single disk

—Each disk is designed for 64 distinct files—more directory entries may be allocated if necessary

—Individual users are physically separated by user numbers, with facilities for file copy operations from one user area to another

—Relative-record random-access functions provide direct access to any of the 65,536 records of an eight-megabyte file

SPECIFICATIONS

Hardware Required

—Model 800 with 720 kit

—DS 235 kit or MDS 225 with 720 kit (integral drive supported except as system boot device)

—iPDS Personal Development System
Optional:
RAM up to 64K
—Additional floppy disk drives
—Single density via 201 controller
—Bubble memory and optional Shugart 460 5¼" disk drive for iPDS

Documentation Package

Title
CP/M 2.2 documentation consisting of 7 manuals:
An Introduction to CP/M Features and Facilities
CP/M 2.2 User's Guide
CP/M Assembler (ASM) User's Guide
CP/M Dynamic Debugging Tool (DDT) User's Guide
ED: A Context Editor for the CP/M Disk System User's Manual
CP/M 2 Interface Guide
CP/M 2 Alteration Guide

Shipping Media

(Specify by Alpha Character when ordering.)

A—single density (IBM 3740/1 compatible)

B—double density

F—double-sided, double density 5¼" floppy (iPDS format)

Order Code	Product Description
See Price List	CP/M (Control Program for Microcomputers) is a disk-based operating system for the Intel 8080/8085-based systems. CP/M provides a general environment for program development, test, execution and storage. CP/M storage is available via a comprehensive, named-file structure supporting both sequential and random access. CP/M support tools include a Text Editor, a debugger, and an 8080/8085 assembler.

SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

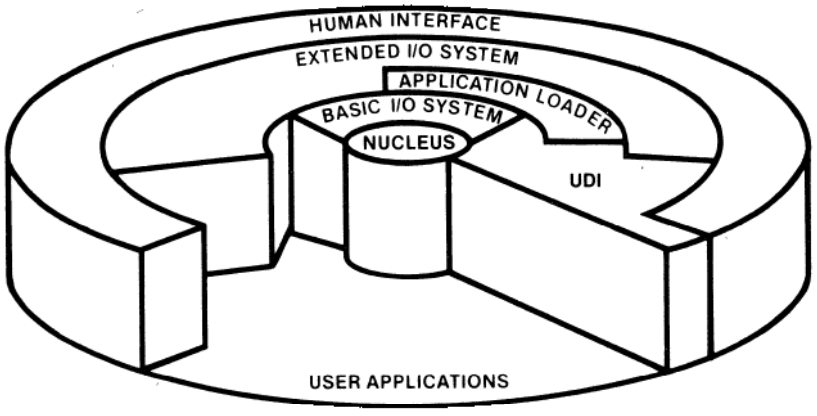
An Intel Software License required.
*CP/M is a registered trademark of Digital Research, Inc.
*CP/M-86, MP/M, CP/NET and MP/NET are trademarks of Digital Research, Inc

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

iRMX™ 86 OPERATING SYSTEM

- **Real-time processor management for time-critical iAPX 86, iAPX 88, iAPX 186, iAPX 188, and iAPX 286 (Real Address Mode) applications**
- **On-target system development with Universal Development Interface (UDI)**
- **Configurable system size and function for diverse application requirements**
- **All iRMX™ 86 code can be (P)ROM'ed to support totally solid state designs**
- **Compatible operating system services for iAPX 86/30, 88/30, 186/30 and 188/30 Operating System Processors (iOSP™ 86)**
- **Configured systems for the iAPX 86 and iAPX 286 processors in Intel integrated system products (iSYS 86/300 and iSYS 286/300)**
- **Multi-terminal support with multi-user human interface**
- **Broad range of device drivers included for industry standard MULTIBUS® peripheral controllers**
- **Complete support of 8087 and 80287 processor extension**
- **Powerful utilities for interactive configuration and real-time debugging**

The iRMX™ 86 Operating System is an easy-to-use, real-time, multi-tasking and multi-programming software system designed to manage and extend the resources of iSBC® 86, iSBC 88, iSBC 186, iSBC 188, and iSBC 286 Single Board Computers, as well as other iAPX 86, iAPX 88, iAPX 186, iAPX 188, and iAPX 286 (Real Address Mode) based microcomputers. iRMX 86 functions are available in silicon with the iAPX 86/30, 88/30, 186/30 and 188/30 Operating System Processors, in a user configurable software package. iRMX 86 functions are also fully integrated into the SYSTEM 86/300 and SYSTEM 286/300 Family of Microcomputer Systems. The Operating System provides a number of standard interfaces that allow iRMX 86 applications to take advantage of industry standard device controllers, hardware components, and a number of software packages developed by Independent Software Vendors (ISVs). Many high-performance features extend the utility of iRMX 86 Systems into applications such as data collection, transaction processing, and process control where immediate access to advances in VLSI technology is paramount. These systems may deliver real-time performance and explicit control over resources; yet also support applications with multiple users needing to simultaneously access terminals. The configurable layers of the System provide services ranging from interrupt management and standard device drivers for many sophisticated controllers, to data file maintenance commands provided by a comprehensive multi-user human interface. By providing access to the standard Universal Development Interface (UDI) for each user terminal, Original Equipment Manufacturers (OEMs) can pass program development and target application customization capabilities to their users.



iRMX™ VLSI Operating System

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1984

April, 1984
Order Number: 210885-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The iRMX 86 Operating System is a complete set of system software modules that provide the resource management functions needed by computer systems. These management functions allow Original Equipment Manufacturers (OEMs) to best use resources available in microcomputer systems while getting their products to market quickly, saving time and money. Engineers are relieved of writing complex system software and can concentrate instead on their application software.

This data sheet describes the major features of the iRMX 86 Operating System. The benefits provided to engineers who write application software and to users who want to take advantage of improving microcomputer price and performance are explained. The first section outlines the system resource management functions of the Operating System and describes several system calls. The second section gives a detailed overview of iRMX 86 features aimed at serving both the iRMX 86 system designer and programmer, as well as the end users of the product into which the Operating System is incorporated.

FUNCTIONAL DESCRIPTION

To take best advantage of iAPX 86, 88, 186, 188, and 286 (Real Address Mode) microprocessors in applications where the computer is required to perform many functions simultaneously, the iRMX 86 Operating System provides a multiprogramming environment in which many independent, multi-tasking application programs may run. The flexibility of independent environments allows application programmers to separately manage each application's resources during both the development and test phases.

The resource management functions of the iRMX 86 System are supported by a number of configurable software layers. While many of the functions supplied by the innermost layer, the Nucleus, are required by all systems, all other functions are optional. The I/O systems, for example, may be omitted in systems having no secondary storage requirement. Each layer provides functions that encourage application programmers to use modular design techniques for quick development of easily maintainable programs.

The components of the iRMX 86 Operating System provide both implicit and explicit management of system resources. These resources include processor scheduling, up to one megabyte of system memory, up to 57 independent interrupt sources, all input and output devices, as well as directory and data files contained on mass storage devices and accessed by a number of independent users. Management of these system resources and methods for sharing resources between multiple processors and users is discussed in the following sections.

Process Management

To implement multi-tasking application systems, programmers require a method of managing the different processes of their application, and for allowing the processes to communicate with each other. The Nucleus layer of the iRMX 86 System provides a number of facilities to efficiently manage these processes, and to effectively communicate between them. These facilities are provided by system calls that manipulate data structures called tasks, jobs, regions, semaphores and mailboxes. The iRMX 86 System refers to these structures as "objects".

Tasks are the basic element of all applications built on the iRMX 86 Operating System. Each task is an entity capable of executing CPU instructions and issuing system calls in order to perform a function. Tasks are characterized by their register values (including those of an optional 8087 or 80287 Numeric Processor Extension), a priority between 0 and 255, and the resources associated with them.

Each iRMX 86 task in the system is scheduled for operation by the iRMX 86 Nucleus. Figure 1 shows the five states in which each task may be placed, and some examples of how a task may move from one state to another. The iRMX 86 Nucleus ensures that each task is placed in the correct state, defined by the events in its external environment and by the task issuing system calls. Each task has a priority to indicate its relative importance and need to respond to its environment. The Nucleus guarantees that the highest priority ready-to-run task is the task that runs.

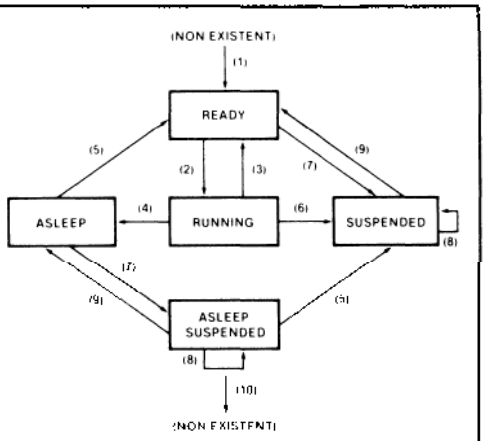
Jobs are used to define the operating environment of a group of tasks. Jobs effectively limit the scope of an application by collecting all of its tasks and other objects into one group. Because the environment for execution of an application is defined by an iRMX 86 job, separate applications can be efficiently developed by separate development teams.

The iRMX 86 Operating System provides two primary techniques for real-time event synchronization in multitask applications: regions and semaphores.

Regions are used to restrict access to critical sections of code and data. Once the iRMX 86 Operating System gives a task access to resources guarded by a region, no other tasks may make use of the resources, and the task is given protection against deletion and suspension. Regions are typically used to protect data structures from being simultaneously updated by multiple tasks.

Semaphores are used to provide mutual exclusion between tasks. They contain abstract "units" that are sent between the tasks, and can be used to implement the cooperative sharing of resources.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



NOTES:
(1) Task is created
(2) Task becomes highest priority ready task
(3) Task gets pre-empted by one with higher priority
(4) Task calls SLEEP or task waits at an exchange
(5) Task sleep period has ended, message was sent to waiting task or wait has ended
(6) Task calls SUSPEND on self
(7) Task suspended by other than self
(8) Task suspended by other than self or a resume that did not bring suspension depth to zero
(9) Task was resumed by other task
(10) Task is deleted

Figure 1. Task State Diagram

Multi-tasking applications must communicate information and share system resources among cooperating tasks. The iRMX 86 Operating System assigns a unique 16-bit number, called a token, to each object created in the System. Any task in possession of this token is able to access the object. The iRMX 86 Nucleus allows tasks to gain access to objects, and hence system resources, at run-time with two additional mechanisms: mailboxes and object directories.

Mailboxes are used by tasks wishing to share objects with other tasks. A task may share an object by sending the object token via a mailbox. The receiving task can check to see if a token is there, or can wait at the mailbox until a token is present.

Object Directories are also used to make an object available to other tasks. An object is made public by cataloging its token and name in a directory. In this manner, any task can gain access to the object by knowing its name, and job environment that contains the directory.

Two example jobs are shown in Figure 2 to demonstrate how two tasks can share an object that was not

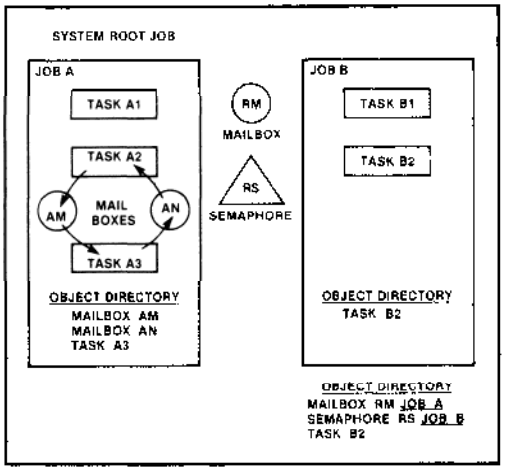


Figure 2. Multiple Jobs Example

known to the programmer at the time the tasks were developed. Both Job 'A' and Job 'B' exist within the environment of the 'Root Job' that forms the foundation of all iRMX 86 systems. Each job possesses a directory in which tasks may catalog the name of an object. Semaphore 'RS', for example, is accessable by all tasks in the system, because its name is cataloged in the directory of the Root Job. Mailbox 'AN' can be used to transfer objects between Tasks 'A2' and 'A3' because its token is accessable in the object directory for Job 'A'.

Table 1 lists the major functions of the iRMX 86 Nucleus that manage system processes.

Memory Management

Each job in an iRMX 86 System defines the amount of the one megabyte of addressable memory to be used by its tasks. The iRMX 86 Operating System manages system memory and allows jobs to share this critical resource by providing another object type: segments.

Segments are contiguous pieces of memory between 16 Bytes and 64K Bytes in length, that exist within the environment of the job in which they were created. Segments form the fundamental piece of system memory used for task stacks, data storage, system buffers, loading programs from secondary storage, passing information between tasks, etc.

The example in Figure 2 also demonstrates when information is shared between Tasks 'A2' and 'A3'; 'A2' only needs to create a segment, put the information in the memory allocated, and send it via the Mailbox 'AM' using the RQ$SEND$MESSAGE system call (see Table 1). Task 'A3' would get the message by using the RQ$RECEIVE$MESSAGE system call. The Figure also shows how the receiving task could signal the sending task by sending an acknowledgement via the second Mailbox 'AN'.

Order Number: 210885-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 1. Process Management System Calls

System Call	Function Performed
RQ$CREATE$JOB	Creates an environment for a number of tasks and other objects, as well as creating an initial task and its stack.
RQ$DELETE$JOB	Deletes a job and all the objects currently defined within its bounds. All memory used is returned to the job from which the deleted job was created.
RQ$OFFSPRING	Provides a list of all the current jobs created by the specified job.
RQ$CATALOG$OBJECT	Enters a name and token for an object into the object directory of a job.
RQ$UNCATALOG$OBJECT	Removes an object's token and its name from a job's object directory
RQ$LOOKUP$OBJECT	Returns a token for the object with the specified name found in the object directory of the specified job.
RQGETTYPE	Returns a code for the type of object referred to by the specified token.
RQ$CREATE$MAILBOX	Creates a mailbox with queues for waiting tasks and objects with FIFO or PRIORITY discipline.
RQ$DELETE$MAILBOX	Deletes a mailbox.
RQ$SEND$MESSAGE	Sends an object to a specified mailbox. If a task is waiting, the object is passed to the appropriate task according to the queuing discipline. If no task is waiting, the object is queued at the mailbox.
RQ$RECEIVE$MESSAGE	Attempts to receive an object token from a specified mailbox. The calling task may choose to wait for a specified number of system time units if no token is available.
RQ$DISABLE$DELETION	Prevents the deletion of a specified object by increasing its disable count by one.
RQ$ENABLE$DELETION	Reduces the disable count of an object by one, and if zero, enables deletion of that object.
RQ$FORCE$DELETE	Forces the deletion of a specified object if the disable count is either 0 or 1.
RQ$CREATE$TASK	Creates a task with the specified priority and stack area.
RQ$DELETE$TASK	Deletes a task from the system, and removes it from any queues in which it may be waiting.
RQ$SUSPEND$TASK	Suspends the operation of a task. If the task is already suspended, its suspension depth is increased by one.
RQ$RESUME$TASK	Resumes a task. If the task had been suspended multiple times, the suspension depth is reduced by one, and it remains suspended.
RQ$SLEEP	Causes a task to enter the ASLEEP state for a specified number of system time units.
RQGETTASK$TOKENS	Gets the token for the calling task or associated objects within its environment.
RQSETPRIORITY	Dynamically alters the priority of the specified task.
RQGETPRIORITY	Obtains the current priority of a specified task.
RQ$CREATE$REGION	Creates a region, with an associated queue of FIFO or PRIORITY ordering discipline.
RQ$DELETE$REGION	Deletes the specified region if it is not currently in use.
RQ$ACCEPT$CONTROL	Gains control of a region only if the region is immediately available.
RQ$RECEIVE$CONTROL	Gains control of a region. The calling task may specify the number of system time units it wishes to wait if the region is not immediately available.
RQ$SEND$CONTROL	Relinquishes control of a region.
RQ$CREATE$SEMAPHORE	Creates a semaphore.
RQ$DELETE$SEMAPHORE	Deletes a semaphore.
RQ$SEND$UNITS	Increases a semaphore counter by the specified number of units.
RQ$RECEIVE$UNITS	Attempts to gain a specified number of units from a semaphore. If the units are not immediately available, the calling task may choose to wait.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Each job is created with both maximum and minimum limits set for its memory pool. Memory required by all objects and resources created in the job is taken from this pool. If more memory is required, a job may be allowed to borrow memory from the pool of its containing job (the job from which it was created). In this manner, initial jobs may efficiently allocate memory to jobs they subsequently create, without knowing their exact requirements.

The iRMX 86 Operating System supplies other memory managment functions to search specific address ranges for available memory. The System performs this search at system initialization, and can be configured to ignore non-existent memory and addresses reserved for I/O devices and other application requirements.

Table 2 lists the major system calls used to manage the system memory.

Interrupt Management

Real-time systems, by their nature, must respond to asynchronous and unpredictable events quickly. The iRMX 86 Operating System uses interrupts and the event-driven Nucleus described earlier to give real-time response to events. Use of a pre-emptive scheduling technique ensures that the servicing of high priority events always takes precedence over other system activities.

The iRMX 86 Operating System gives applications the flexibility to optimize either interrupt response time or interrupt response capability by providing two tiers of Interrupt Management. These two distinct tiers are managed by Interrupt Handlers and Interrupt Tasks.

Interrupt Handlers are the first tier of interrupt service. For small simple functions, interrupt handlers are often the most efficient means of responding to an event. They provide faster response than interrupt tasks, but must be kept simple since interrupts (except the iAPX 86, 88, 186, 188, and 286 non-maskable interrupt) are masked during their execution. When extended service is required, interrupt handlers "signal" a waiting interrupt task that, in turn, performs more complicated functions.

Interrupt Tasks are distinct tasks whose priority is associated with a hardware interrupt level. They are permitted to make any iRMX 86 system call. While an interrupt task is servicing an interrupt, interrupts of lower priority are not allowed to pre-empt the system.

Table 3 shows the iRMX 86 System Calls provided to manage interrupts.

Table 2. Memory Management System Calls

System Call	Function Performed
RQ$CREATE$SEGMENT	Dynamically allocates a memory segment of the specified size.
RQ$DELETE$SEGMENT	Deletes the specified segment by deallocating the memory.
RQGETPOOL$ATTRIBUTES	Returns attributes such as the minimum and maximum, as well as current size of the memory in the environment of the calling task's job.
RQGETSIZE	Returns the size (in bytes) of a segment.
RQSETPOOL$MIN	Dynamically changes the minimum memory requirements of the job environment containing the calling task.

Table 3. Interrupt Management System Calls

System Call	Function Performed
RQSETINTERRUPT	Assigns an interrupt handler and, if desired, an interrupt task to the specified interrupt level. Usually the calling task becomes the interrupt task.
RQ$RESET$INTERRUPT	Disables an interrupt level, and cancels the assignment of the interrupt handler for that level. If an interrupt task was assigned, it is deleted
RQGETLEVEL	Returns the number of the highest priority interrupt level currently being processed.
RQ$SIGNAL$INTERRUPT	Used by an interrupt handler to signal the associated interrupt task that an interrupt has occurred
RQ$WAIT$INTERRUPT	Used by an interrupt task to SLEEP until the associated interrupt handler signals the occurrence of an interrupt.
RQ$EXIT$INTERRUPT	Used by an interrupt handler to relinquish control of the System.
RQ$ENABLE	Enables the hardware to accept interrupts from a specified level
RQ$DISABLE	Disables the hardware from accepting interrupts at or below a specified level.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INTERRUPT MANAGEMENT EXAMPLE

Figure 3 illustrates how the iRMX 86 Interrupt System may be used to output strings of characters to a printer. In the example, a mailbox named 'PRINT' is used by all tasks in the system to queue messages to be printed. Application tasks put the characters in segments that are transmitted to the printer interrupt task via the PRINT Mailbox. Once printing is complete, the same interrupt task passes the messages on to another application via the FINISHED Mailbox so that an operator message can be displayed.

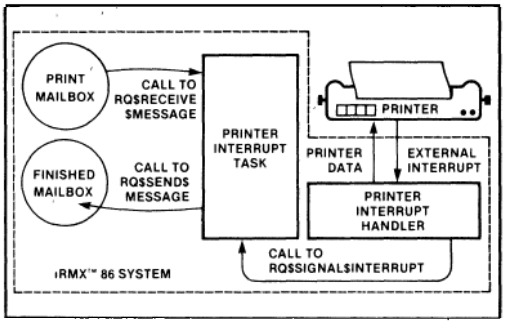


Figure 3. Interrupt Management Example

Basic I/O System

The Basic I/O System (BIOS) provides the direct access to I/O devices needed by real-time applications. The BIOS allows I/O functions to overlap other system functions. In this manner, application tasks make asynchronous calls to the iRMX 86 BIOS, and proceed to perform other activities. When the I/O request must be completed before an application can continue, the task waits at a mailbox for the result of the operation. Some system calls provided by the BIOS are listed in Table 4.

The Basic I/O System communicates with peripheral devices through device drivers. These device drivers provide the System with four basic functions needed to control and communicate with devices: Initialize I/O, Finish I/O, Queue I/O, and Cancel I/O. Using the device driver interface, users of non-standard devices may write custom drivers compatible with the I/O System.

The iRMX 86 Operating System includes a number of device drivers to allow applications to use standard USART serial communications devices, multiple CRTs and keyboards, bubble memories, diskettes, disks, a Centronics-type parallel printer, and many of Intel's iSBC and iSBX™ device controllers (see Table 8). If an application requires use of a non-standard device, users need only write a device driver to be included with the BIOS, and access it as if it were part of the standard system. For most common random-access devices, this job is further simplified by using standard routines provided with the System. Use of this technique ensures that applications can remain device independent.

Multi-Terminal Support

The iRMX 86 Terminal Support provides line editing and terminal control capabilities. The Terminal Support communicates with devices through simple drivers that do only character I/O functions. Dynamic terminal reconfiguration is provided so that attributes such as terminal type and line speed may be changed without modifying the application or the Operating System. Dynamic configuration may be typed in, generated programmatically or stored in a file and copied to a terminal I/O connection.

Table 4. Key BIOS I/O Management System Calls

System Call	Function Performed
RQAATTACH$FILE	Creates a Connection to an existing file
RQACHANGE$ACCESS	Changes the types of accesses permitted to the specified user(s) for a specific file
RQACLOSE	Closes the Connection to the specified file so that it may be used again, or so that the type of access may be changed.
RQACREATE$DIRECTORY	Creates a Named File used to store the names and locations of other Named Files
RQACREATE$FILE	Creates a data file with the specified access rights
RQADELETE$CONNECTION	Deletes the Connection to the specified file
RQAGETSFILE$STATUS	Returns the current status of a specified file.
RQAOPEN	Opens a file for either read, write, or update access.
RQAREAD	Reads a number of bytes from the current position in a specified file
RQASEEK	Moves the current data pointer of a Named or Physical file
RQAWRITE	Writes a number of bytes at the current position in a file
RQ$WAIT$IO	Synchronizes a task with the I/O System by causing it to wait for I/O operation results

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The iRMX 86 Terminal Support provides automatic translation of control characters to specific control sequences for each terminal. This translation enables applications using standard control characters to function with non-standard terminals. The translation requirements for each terminal can be stored in terminal description files and copied to a connection, as described above.

Disk I/O Performance

Figure 4 shows iRMX 86 performance obtained using the iSBC 215 Winchester Disk and iSBX 218A Diskette Controllers under the specified conditions. The vertical axis is a linear scale of throughput in units of 10,000 bytes per second. The horizontal axis is a logrithmic scale showing the transfer size for the reads and writes. Each data point on the graph indicates the time required for a read/write request of 64K bytes. Therefore each transfer size on the horizontal scale less than 64K was repeated until a total request of 64K was read or written.

Each device driver can be used to interface to a number of separate and, in some cases, different devices (see Figure 5). The iSBC 215 Device Driver, supplied with the system, is capable of supporting the iSBC 215 Winchester Disk Controller, the iSBC 220 SMD Disk Controller, and the iSBX 218A Flexible Disk Controller (when mounted on an iSBC 215 board). Each device controller may, in turn, control a number of separate device units. In addition, each driver may control a number of like device controllers. This capability allows the use of large storage systems with a minimum of I/O system code to write or maintain.

Extended I/O System

The iRMX 86 Extended I/O System (EIOS) adds a number of I/O management capabilities to simplify access to files. Whereas the BIOS provides users with the basic system calls needed for direct management of I/O resources, many users prefer to have the system perform all the buffering and synchronization of I/O requests automatically. The EIOS allows users to access I/O devices without having to write procedures for buffering data, or to specify particular devices with constant device names.

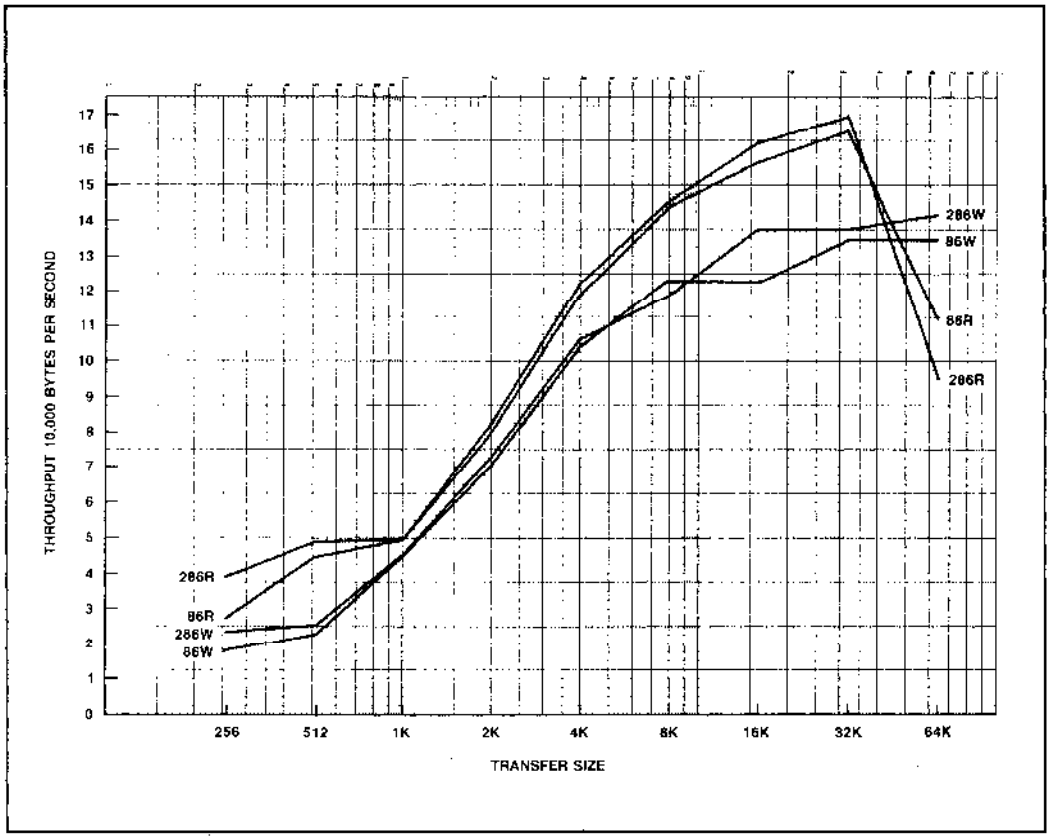


Figure 4. iRMX™ 86 Disk I/O Performance

Order Number: 210885-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

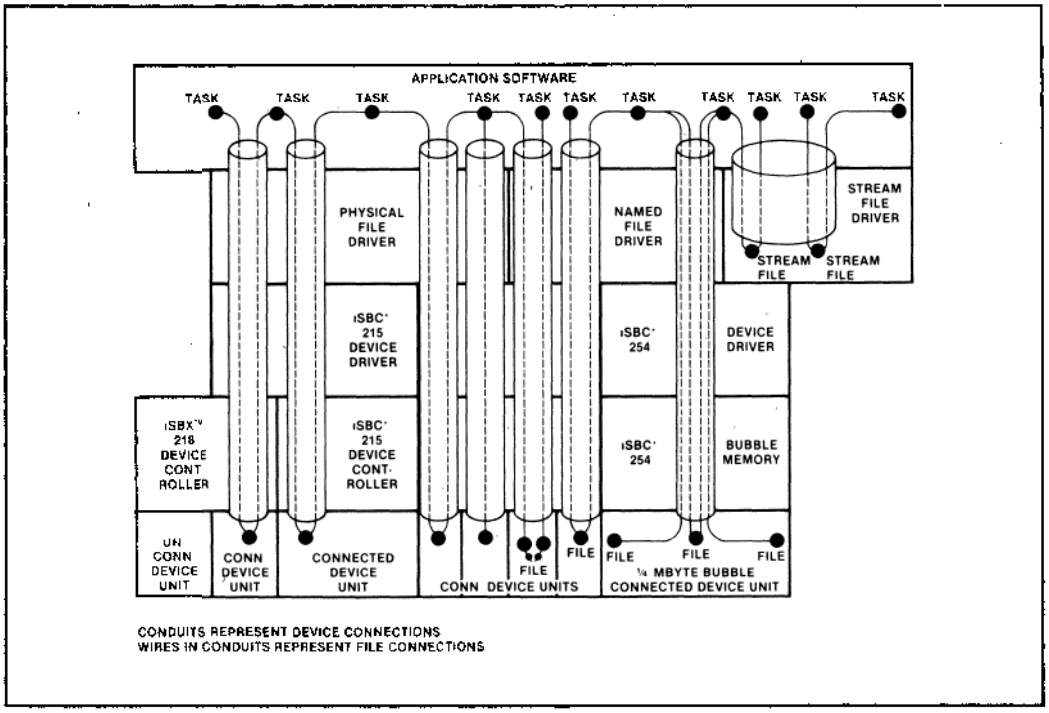


Figure 5. Device Driver and Controller Relationships

By performing device buffering automatically, the iRMX 86 EIOS optimizes accesses to disks and other devices. Often, when an application task asks the System to READ a portion of a file, the System is able to respond immediately with the data it has read in advance of the request. Similarly, the EIOS will not delay a task for writing data to a device unless it is specifically told to, or if its output buffers are filled.

Logical file and device names are provided by the EIOS to give applications complete file and device independence. Applications may send data to the 'line printer' (:LP:) without needing to know which specific device will be used as the printer. This logical name may, in fact, not be a printer at all, but it could be a disk file that is later scheduled for printing.

The EIOS uses the functions provided by the BIOS to synchronize individual I/O requests with results returned by device drivers. Most EIOS system calls are similar to the BIOS calls, except that they appear to suspend the operation of the calling task until the I/O requests are completed.

Two new primitives have been added to the EIOS. These are: RQ$HYBRID$DETACH$DEVICE and RQ$GET$LOGICAL$DEVICE$STATUS.

RQ$HYBRID$DETACH$DEVICE allows a programmer to temporarily detach a device physically so it can be temporarily attached another way.

RQGETLOGICAL$DEVICE$STATUS provides information about a logical device: the physical device name, file driver, number of connections to the device, and the owner of the device.

File Management

The iRMX 86 Operating System provides three distinct types of files to ensure efficient management of both program and data files: Named Files, Physical Files, and Stream Files. Each file type provides access to I/O devices through the standard device drivers mentioned earlier. The same device driver is used to access physical and named files for a given device.

NAMED FILES

Named files allow users to access information on secondary storage by referring to a file with its ASCII name. The names of files stored on a device are stored in special files called directories. As directories are themselves named files, the iRMX 86 File System allows directories to contain the names of other directories. Figure 6 illustrates the resulting hierarchical file structure. This structure is useful for isolating file names to particular user applications, and for tailoring system data to the requirements of users and applications sharing storage devices. Using different branches on the directory tree, different users do not have to coordinate in naming their files to ensure unique names.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

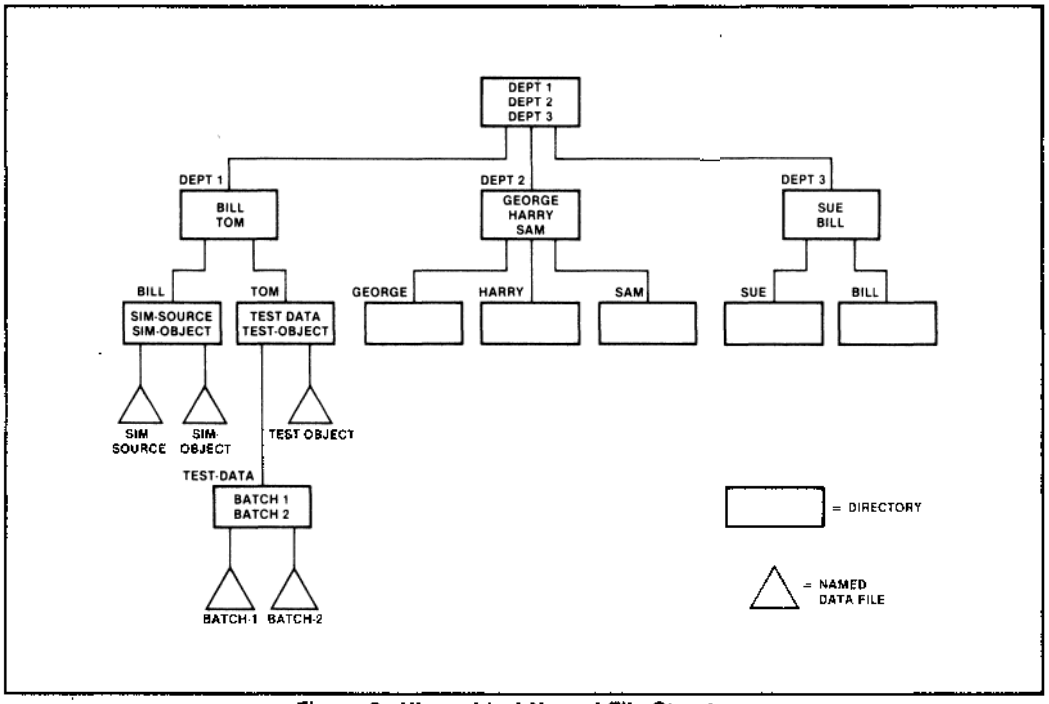


Figure 6. Hierarchical Named File Structure

Whenever a request is made involving a file name, the System will search the appropriate directory in order to find the necessary information about the file's size, access rights, and specific location on the storage device.

The iRMX 86 BIOS uses an efficient format for writing the directory and data information into secondary storage. This standard iRMX 86 format is fully compatible with the ISO Media standard, and other Intel systems such as the iRMX 88 Operating System. This structure enables the system to directly access any byte in a file, often without having to do additional I/O to access space allocation information. The maximum size of an individual file is 4.3 billion bytes.

EASE OF ACCESS

The hierarchical file structure is provided to isolate and organize collections of named files. To give operators fast and simple access to any level within the file tree, an ATTACHFILE command is provided. This command allows operators to create a logical name to a point in the tree so that a long sequence of characters need not be typed each time a file is referred to.

ACCESS PROTECTION

Access to each Named File is protected by the rights assigned to each user by the owner of the file. Rights to read, append, update, and delete may be selectively granted to other users of the system. In general, users of Named Files are classified into one of two categories: User and World. Users are used when different programmers and programs need to share information stored in a file. The World classification is used when rights are to be granted to all who can use the system.

PHYSICAL FILES

Physical Files allow more direct device access than Named Files. Each Physical File occupies an entire device, treated as a single stream of individually accessable bytes. No access control is provided for Physical Files as they are typically used for such applications as driving a printing device, translating from one device format to another, driving a paper tape device, real-time data acquisition, and controlling analog mechanisms.

STREAM FILES

Stream Files provide applications with a method of using iRMX 86 file management methods for data that does not need to go into secondary storage. Stream Files act as direct channels, through system memory, from one task to another. These channels are very useful to programs, for example, wishing to preserve file and device independence allowing data sent to a printer one time, to a disk file another time, and to another program on a different occasion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BOOTSTRAP AND APPLICATION LOADERS

Two utilities are supplied with the System to load programs and data into system memory from secondary storage devices:

The iRMX 86 Bootstrap Loader can be configured to a size of less than 1K Bytes of P(ROM), and is typically used to load the initial system from the system disk into memory, and begin its execution. Error reporting and debug switch features have been added to the Bootstrap Loader. When the Bootstrap Loader detects errors such as: file does not exist or device not ready, an error message is reported back to the user. The debug switch will cause the Bootstrap Loader to load the system but not begin its execution. Instead the Bootstrap Loader will pass control to the monitor at the first instruction to be executed by the system.

The Application Loader is typically used by application programs already running in the system to load additional programs and data from any secondary storage device. The Human Interface layer, for example, uses the Application Loader to load the non-resident Human Interface Commands. The Application Loader is capable of loading both relocatable and absolute code as well as program overlays.

Human Interface

The flexibility of the interface between computer controlled machines and their users often determines the usability and ultimate success of the machines. Table 11 lists iRMX 86 Human Interface functions giving users and applications simple access to the file and system management capabilities described earlier. The process, interrupt, and memory managment functions described earlier, are performed automatically for Human Interface users.

MULTI-USER ACCESS

Using the multi-terminal support provided by the BIOS, the iRMX 86 Human Interface can support several simultaneous users. The real-time nature of the system is maintained by providing a priority for each user, and using the event-driven iRMX 86 Nucleus to schedule tasks. High-performance interrupt response is guaranteed even while users interact with various application packages. For example, multi-terminal support allows one person to be using the iRMX 86 Editor, while another compiles a FORTRAN 86 or PASCAL 86 program, while several others load and access applications.

Each terminal attached to the iRMX 86 multi-user Human Interface is automatically associated with a user, a memory pool, and an initial program to run when the terminal is connected. This association is made using a file that may be changed at any time. Changes are effective the next time the system is initialized.

The initial program specified for each terminal can be a special application program, a custom Human Interface, or the standard iRMX 86 Command Line Interpreter (CLI). For example, you may choose to use the Microsoft Basic Interpreter as this initial program. After system start-up, each terminal user would be able to run the interpreter without asking for it to be loaded. From the BASIC interpreter, an operator, for example, could run a data collection program, written in BASIC, that communicates with several laboratory instruments, and prints charts and reports based on certain test results. When finished entering, changing, or running a BASIC program, the terminal would remain in BASIC for the next user.

Specifying an application program as a terminal's initial program makes the interface between operators and the computer system much simpler. Each operator need only be aware of the function of a particular application; not needing to interact with any unfamiliar functions also available on the application system.

Specifying the standard iRMX 86 Human Interface CLI as the initial program enables users of the terminals to access all iRMX 86 functions. This CLI makes it easy to manage iRMX 86 files, load and execute Intel-supplied and custom programs, and submit command files for execution.

FEATURE OVERVIEW

The iRMX 86 Operating System is well suited to serve the demanding needs of real-time applications executing on complex microprocessor systems. The iRMX 86 System also provides many tools and features needed by real-time system developers and programmers. The following sections describe features useful in both the development and execution environments. The description of each feature outlines the advantages given to hardware and software engineers concerned with overall system cost, expandability with custom and industry standard options, and long-term maintenance of iRMX 86-based systems. The development environment features also describe the ease with which the iRMX 86 Operating System can be incorporated into overall system designs.

Execution Environment Features

REAL-TIME PERFORMANCE

The iRMX 86 Operating System is designed to offer the high performance, multi-tasking functions required by real-time systems. Designers can make use of the latest VLSI devices such as the 8087 or 80287 Numeric Processor Extension, and the 80130 Operating System Firmware Component to improve their system cost/performance ratio or the iMMX™ 800 MULTIBUS® Message Exchange software package to divide and coordinate various system activities among multiple processors. Typical iRMX 86 system performance characteristics are shown in Table 5.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Many real-time systems require high performance operation. To meet this requirement, all of iRMX 86 can be put into zero wait-state P(ROM). This approach eliminates the possibility of disk access times slowing down performance, while allowing system designers to take advantage of high performance memory devices.

CONFIGURABILITY

The iRMX 86 Operating System is configurable by system layer, and by system call within each layer. In addition all the I/O port addresses used by the System are configurable by the user. This flexibility gives designers the freedom to choose configurations of hardware and software that best suit their size and functional requirements. Two example configurations are shown in Figure 7.

Table 5. iRMX™ Real-Time Performance Using iSBC® 86/30 and iSBC® 286/10 Single Board Computers

Real-Time Function	iSBC® 86/30 Execution Time (msec)	iSBC® 286/10 Execution Time (msec)
Suspend Task	1.02	0.83
Interrupt Latency (to handler)	0.29 (Max)	0.20 (Max)
Interrupt Latency (to handler)	0.02 (Typical)	0.03 (Typical)
Context Switch Caused By Interrupt	0.84 (Max)	0.78 (Max)
Send Message (no context switch)	0.32	0.25
Send Message (with context switch)	0.58	0.49
Send Control (no context switch)	0.21	0.16
Send Control (with context switch)	0.64	0.54
Receive Control (no waiting)	0.26	0.19

Context switch time is the time between executing in the context of a task, and the first instruction to execute in the context of another task.

The execution times shown in Column 2 were measured using an 8MHz iSBC Single Board Computer, 256K on-board RAM, and all program and data stored in on-board RAM.

The execution times shown in Column 3 were measured using a 5MHz iSBC 286/10 Single Board Computer, no on-board RAM, and all program and data stored in LBX RAM.

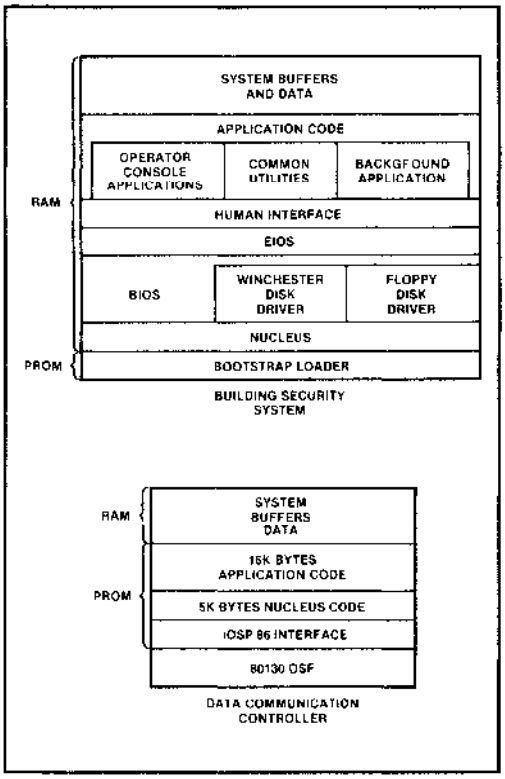


Figure 7. Typical iRMX™ 86 Configurations

Most configuration options are selected during system design stages. Others may be selected during system operation. For example, the amount of memory devoted to queues within a Mailbox can be specified at the time the Mailbox is created. Devoting more memory to the Mailbox allows more messages to be transmitted to other tasks without having to degrade system performance to allocate additional memory dynamically.

The chart shown in Table 6 indicates the actual memory size required to support these different configurations of the iRMX 86 System. Systems requiring only Nucleus level functions may require no more than 13K bytes for the Operating System. (Use of the iAPX 86/30 requires only 4K bytes of RAM, 7K bytes of initialization code in EPROM and the 16K bytes of code in the 80130.) Other applications, needing I/O management functions, may select portions of additional layers that fit their needs and size constraints.

This configurability also applies to the Terminal Handler, Dynamic Debugger, and System Debugger. The Terminal Handler provides a serial terminal interface in a system that otherwise doesn't need an I/O system. Either one of the debuggers need to be included only as debugging tools (usually only during system development).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 6. iRMX™ 86 Configuration Size Chart

System Layer	Min. ROMable Size	Max. Size	Data Size
Bootstrap Loader	1K	1.5K	6K*
Nucleus	10.5K	24K	2K
BIOS	26K	78K	1K
Application Loader	4K	10K	2K
EIOS	10.5K	12.5K	1K
Human Interface	22K	22K	15K
UDI	8K	8K	0
Terminal Handler	3K	3K	0.3K
System Debugger	20K	20K	1K
Dynamic Debugger	28.5K	28.5K	1K
Human Interface Commands			116K
Interactive Configuration Utility			308K

* Usable by System after bootloading

MULTI-PROCESSING

The resources provided by a single processor are often not enough to perform certain functions. With the standard interfaces provided by the iMMX 800 MULTIBUS Message Exchange package, the iRMX 86 Operating System supports a loosely-coupled multi-processing environment. Task running on one processor may communicate with tasks running on other processors, even if they operate under different operating systems. The iMMX 800 software is capable of sending messages over the MULTIBUS to tasks operating under either the iRMX 88 Executive, or the iRMX 86 Operating System. Using this message exchange mechanism, applications may increase their system performance quite easily, improve overall interrupt response, gain access to the iSBC® 550 Ethernet Controller, and leave room for future product enhancements.

MULTI-USER ACCESS

Many real-time systems must provide a variety of users access to system control functions and collected data. The iRMX 86 System provides easy-to-use support for applications to access multiple terminals. It also enables multiple and different users to access different applications concurrently.

Figure 8 illustrates a typical iRMX 86 application simultaneously supporting multi-terminal data collection and real-time environments. Shown is a group of terminals used by machinists on a shop floor to communicate with a job management program, a building security system that constantly monitors energy usage requirements, a system operator console capable of accessing all system functions, and a group of terminals in the Production Engineering department used to monitor job costs while developing new device control specifications instructions. The iSBC 544 Intelligent Terminal Interface supports multiple user terminals without degrading system performance to handle character I/O.

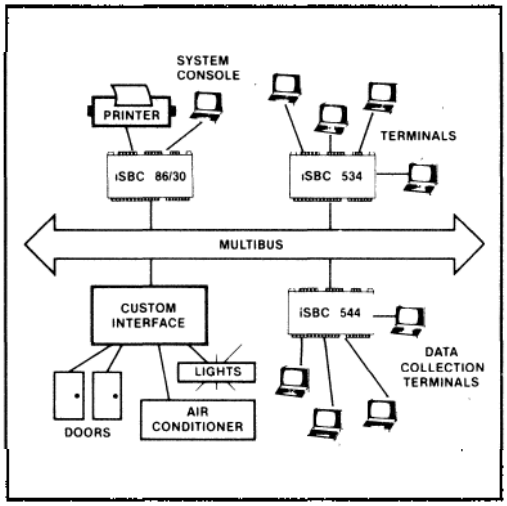


Figure 8. Multi-Terminal and Multi-User Real-Time System

EXTENDABILITY

The iRMX 86 Operating System provides three means of extensions. This extendability is essential for support of OEM and volume end user value added features. This ability is provided by: user-defined operating system calls, user-defined objects (similar to Jobs, Tasks, etc.), and the ability to add functions later in the product life cycle. The modular, layered structure of the System easily facilitates later additions to iRMX 86 applications. User-defined objects are supported by the functions listed in Table 7.

Using standard iRMX 86 system calls, users may define custom objects, enabling applications to easily manipulate commonly used structures as if they were part of the original operating system.

Order Number: 210885-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 7. User Extension System Calls

System Call	Function Performed
RQ$CREATE$COMPOSITE	Creates a custom object built of previously defined objects.
RQ$DELETE$COMPOSITE	Deletes the custom object, but not the various objects from which it was built
RQ$INSPECT$COMPOSITE	Returns a list of Token Identifiers for the component objects from which the specified composite object is built.
RQ$ALTER$COMPOSITE	Replaces a component object of a composite object.
RQ$CREATE$EXTENSION	Creates a new type of object and assigns a mailbox used for collecting these objects when they are deleted.
RQ$DELETE$EXTENSION	Deletes an extension definition.

EXCEPTION HANDLING

The System includes predefined exception handlers for typical I/O and parameter error conditions. The error handling mechanism is both configurable and extendable.

SUPPORT OF STANDARDS

The iRMX 86 Operating System supports the many hardware and software standards needed by most application systems to ensure that commonly available hardware and software packages may be interfaced with a minimum of cost and effort. The iRMX 86 System supports the iSBC family of products built on the Intel MULTIBUS (IEEE Standard 796), and a number of standard software interfaces such as the UDI and the common device driver interface (See Figure 9). The procedural interfaces of the UDI are listed in Table 9.

The Operating System includes support for the proposed IEEE 80-bit extended real-variable format of the 8087 Numeric Data Processor, and the IEEE 796 (MULTIBUS) hardware interface. Other standards such as the iMMX 800 MULTIBUS Message Exchange, and an Ethernet communication interface are supported by optional software packages available to run on the iRMX 86 System.

SPECTRUM OF CPU PERFORMANCE

The iRMX 86 Operating System supports a broad range of Intel processors. In addition to support for iAPX 86 and 88 based systems, the iRMX 86 system has been enhanced to support iAPX 186, 188, and 286 (Real Address Mode)-based Systems. This new support enables the user to take advantage of the faster speed and higher performance of Intel's 286 based microprocessors such as the iSBC 286/10 single board computer. By choosing the appropriate CPU, designers can choose from a wide range of performance options, without having to change application software.

COMPONENT LEVEL SUPPORT

The iRMX 86 System may be tailored to support specific hardware configurations. In addition to system memory,

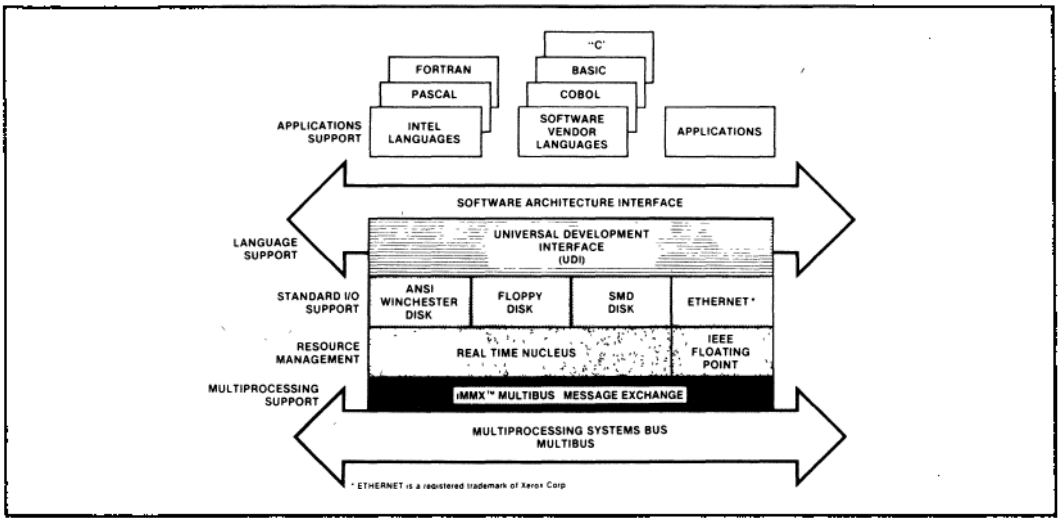


Figure 9. iRMX™ 86 Standard Interfaces

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

only an iAPX 86, iAPX 88, iAPX 186, iAPX 188, or iAPX 286 microprocessor, an 8259A Programmable Interrupt Controller (PIC), and either an 8253, 8274, or 82530 Programmable Interval Timer (PIT) are required as follows:

- iAPX 86 and iAPX 88 systems need either:
  - 8253 PIT and 8259A PIC (master) or
  - 80130 firmware (PIC is master)
- iAPX 186 and iAPX 188 systems where 186 PIC is slave, needs either:
  - 8253 PIT and 8259A PIC (master) or
  - 80130 firmware (PIC is master)

  where 186 PIC is master:
  - Uses 186 PIT for the system clock; no external PIT is needed
  - Can use either
    186 PIC (master) only or
    8259A/80130 PIC (slave)
- iAPX 286 systems need
  - 8253 PIT and 8259A PIC.

Alternatively, the iRMX 86 Operating System may be used in conjunction with the 80130 Operating System Firmware Component that not only provides these hardware functions, but eliminates the need for approximately 16K bytes of the iRMX 86 Nucleus code (see Figure 7). For systems requiring extended mathematics capability, an 8087 or 80287 Numeric Data Processor may be added to perform these functions up to 100 times faster than equivalent software. For applications servicing more than 8 interrupt sources, additional 8259A's may be configured as slave controllers.

BOARD LEVEL SUPPORT

The iRMX 86 Operating System includes device drivers to support a broad range of MULTIBUS device controllers. The particular boards and types of devices supported are listed in Table 8. The device controllers all adhere to industry standard electrical and functional interfaces.

In addition to the on-CPU board terminal drivers, the iRMX 86 BIOS includes two iSBC board-level device drivers to support multiple terminal interfaces:

The iSBC 544 Intelligent Four-Channel Terminal Interface Device Driver provides support for multiple controllers each supporting up to four standard RS232 terminals. The iSBC 544 driver takes advantage of an on-board 8085 processor to greatly reduce the system processor time required for terminal I/O by locally managing input and output buffers. The iSBC 544 firmware provided with the operating system can offload the system CPU by as much as 75% when doing character outputting.

The iSBC 534 Four-Channel USART Controller Device Driver also provides support for multiple controller boards each supporting up to four standard RS232 terminals.

The new RAM disk feature in iRMX 86 makes a portion of the memory address space look like a disk drive to the I/O system.

Table 8. Supported Devices

iSBC® Device Controller	Description
iSBC® 86,88	Serial Port to CRT, Parallel Port to Centronics-type Printer, Interval Timer and Interrupt Controller.
iSBC® 186/03	Small Computer System Interface (SCSI) Supporting All Random Access "Extended Standard" SCSI/SASI hard disk controllers.
iSBC® 204	Single Density Diskette.
iSBC® 206	Cartridge-Type Hard Disk.
iSBC® 208	Single & Double Density, Single & Double Sided, 8" & 5.25" Diskettes.
iSBC® 215(G)	Standard Winchester Disks.
iSBX® 218	Single or Double density, Single or double sided, 8-inch diskettes (when used on an iSBC 215(G)).
iSBX® 218A	Single or Double Density, Single or Double Sided, 8" & 5.25" Diskette (when used on an iSBC 215G Winchester Controller).
iSBC® 220	Standard Storage Module Board.
iSBX® 251	Bubble Memory Multimodule Board.
iSBC® 254(S)	Bubble Memory Board.
iSBX® 351	1-Channel Serial Port to CRTs, Modems.
iSBC® 534,544	4-Channel Serial Ports to CRTs, Modems.
iSBX™ 270	Black and White CRTs and full ASCII keyboards.

NOTE: (G) = optional iSBC 215, iSBC 215B, or iSBC 215G
(S) = optional iSBC 254 or iSBC 254S

Development Environment Features

The iRMX 86 Operating System supports the efficient utilization of programming time by providing important tools for program development. Some of the tools necessary to develop and debug real-time systems are included with the Operating System. Others, such as language compilers, are available from Intel and from leading Independent Software Vendors.

LANGUAGES

The iRMX 86 Operating System supports 31 standard system calls known as the Universal Development Interface (UDI). Figure 9 shows the iRMX 86 standard interfaces to many compilers and language translators, including the iAPX 86 and 88 Macro Assembler; the PASCAL 86/88, PL/M 86/88, FORTRAN 86/88 and C86 compilers available from Intel. Also included are other

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel development tools, language translators and utilities available from other vendors. Any application that ran on the iRMX 86 Release 5 Universal Runtime Interface (URI) will run on the iRMX 86 Release 6 UDI. The full set of UDI calls (which includes the URI system calls) is required to run a compiler.

These standard software interfaces (the UDI) ensure that users of the iRMX 86 Operating System may transport their applications to future releases of the iRMX 86 Operating System and other Intel and independent vendor software products. The calls available in the UDI are shown in Table 9.

Table 9. UDI System Calls

System Call	Function Performed
Memory Management:	
DQ$ALLOCATE	Creates a Segment of a specified size
DQ$FREE	Returns the specified segment to the System
DQGETSIZE*	Returns the size of the specified Segment.
DQ$RESERVE$IO$MEMORY*	Reserves memory to OPEN and ATTACH files
File Management:	
DQ$ATTACH	Creates a Connection to a specified file.
DQ$CHANGE$ACCESS*	Changes the user access rights associated with a file or directory.
DQ$CHANGE$EXTENSION	Changes the extension of a file name in memory.
DQ$CLOSE	Closes the specified file Connection.
DQ$CREATE	Creates a Named File.
DQ$DELETE	Deletes a Named File.
DQ$DETACH	Closes a Named File and deletes its Connection
DQ$OPEN	Opens a file for a particular type of access.
DQGETCONNECTION$STATUS*	Returns the current status of the specified file Connection
DQ$FILE$INFO*	Returns data about a file Connection.
DQ$READ	Reads the next sequence of bytes from a file.
DQ$RENAME*	Renames the specified Named File.
DQ$SEEK	Moves the position pointer of a file.
DQ$TRUNCATE	Truncates a file.
DQ$WRITE	Writes a sequence of bytes to a file.
Process Management:	
DQ$EXIT	Exits from the current application job.
DQ$OVERLAY*	Causes the specified overlay to be loaded
DQ$SPECIAL	Performs special I/O related functions on terminals with special control features.
DQ$TRAP$CC	Captures control when CNTRL/C is typed.
Exception Handling:	
DQGETEXCEPTION$HANDLER	Returns a pointer to the program currently being used to process errors.
DQ$DECODE$EXCEPTION	Returns a short description of the specified error code.
DQ$TRAP$EXCEPTION	Identifies a custom exception processing program for a particular type of error.
Application Assistance:	
DQ$DECODE$TIME	Returns system time and date in binary and ASCII character format.
DQGETARGUMENT*	Returns the next argument from the character string used to invoke the application program.
DQGETSYSTEM$ID*	Returns the name of the underlying operating system supporting the UDI.
DQGETTIME*	Returns the current time of day as kept by the underlying operating system
DQ$SWITCH$BUFFER	Selects a new buffer from which to process commands.

* Calls available only through the UDI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The high performance of the iRMX 86 Operating System enhances the throughput of compilers and other development utilities. Table 10 indicates the average performance of typical development environment functions operating in the same configuration described in Figure 4.

Table 10. Development Environment Performance

Function	Average Execution Time
Directory Command (S Format with 25 files)	5.3 sec
Load the COPY Command	1.2 sec
Copy a 1K Byte File (Winchester to Winchester)	1.0 sec
Copy a 16K Byte File	1.7 sec
Copy a 64K Byte File	3.9 sec
Copy a 1K Byte File (Winchester to Diskette)	1.4 sec
Compile PL/M 86	393 lpm
Compile PASCAL 86 Program	453 lpm

TOOLS

Certain tools are necessary for the development of microcomputer applications. The iRMX 86 Human Interface includes many of these tools as non-resident commands. They can be included on the system disk of a application system, and brought into memory when needed to perform functions as listed in Table 11.

Table 11. Major Human Interface Utilities

Command	Function
BACKUP	Copy directories and files from one device to another.
COPY	Copy one or more files to one or more destination files.
CREATEDIR	Create a directory file to store the names of other files
DIR	List the names, sizes, owners, etc. of the files contained in a directory.
ATTACHFILE	Give a logical name to a specified location in a file directory tree.
PERMIT	Grant or rescind user access to a file.
RENAME	Change the name of a file
SUBMIT	Start the processing of a series of commands stored in a file.
SUPER	Change operator's ID to that of the System Manager with global access rights and privileges.

Table 11. Major Human Interface Utilities (Con.t.)

Command	Function
TIME	Set the system time-of-day clock.
VERIFY	Verify the structure of an iRMX™ 86 Named File volume, and check for possible disk data errors.

INTERACTIVE CONFIGURATION UTILITY

The iRMX 86 Operating System is designed to provide OEMs the ability to configure for specific system hardware and software requirements. The Interactive Configuration Utility (ICU) builds iRMX 86 configurations by asking appropriate questions and making reasonable assumptions. It runs on either an Intellec® Series III development system or iRMX 86 development system that includes a hard disk and the UDI. Table 12 lists the hardware and support software requirements of different iRMX 86 development system environments.

Table 12. iRMX™ Development Environment

Intellec® Series III: MDS 313 PL/M 86/88 Compiler One hard disk and one diskette drive
iRMX™ 86 Development System iRMX™ 860 ASM 86 Assembler and Utilities iRMX™863 PL/M 86/88 Compiler iSDM 86 or 286 System Debug Monitor 512K Bytes of RAM 5M Byte On-Line Storage and one double-density diskette drive
SYSTEM 86/300 or 286/300 Series Microcomputer System Basic configuration

Figure 10 shows one of the many screens displayed during the process of defining a configuration. It shows the abbreviations for each choice on the left, a more complete description with the range of possible answers in the center, and the current (sometimes default) choice on the right. The bottom of the screen shows three changes made by the operator (lower case lettering), and a request for help on the Exception Mode question. In response to a request for help, the ICU displays an additional screen outlining possible choices and some overall system effects.

The ICU requests only information required as a result of previous choices. For example, if no Extended I/O System functions are required, the ICU will not ask any further questions about the EIOS. Once a configuration session is complete, the operator may save all the information in a file. Later when small changes are necessary, this file can be modified. A completely new session is not required.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

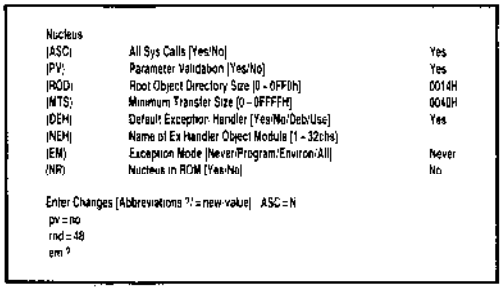


Figure 10. ICU Screen for iRMX™ 86 Nucleus

REAL-TIME DEBUGGING TOOLS

The iRMX 86 Operating System supports three distinct debugging environments: Static, Dynamic, and Post-Mortem. While the iRMX 86 Operating System does support a multi-user Human Interface, these real-time debugging aids are usually most useful in a single-user environment where modifications made to the system cannot affect other users.

System Debugger

The static debugging aid is the iRMX 86 System Debugger. This debugger is an extension of the iSDM 86 and the iSDM 286 System Debug Monitors. The System Debugger provides static debugging facilities when the system hangs or crashes, when the Nucleus is inadvertently overwritten or destroyed, or when synchronization requirements prevent the debugging of certain tasks. The System Debugger stops the system and allow you to examine the state of the system at that instant, and allows you to:

— Identify and interpret iRMX 86 system calls.
— Display information about iRMX 86 objects.
— Examine a task's stack to determine system call history.

iRMX™ 86 Dynamic Debugger

The iRMX 86 Dynamic Debugger runs as part of an iRMX 86 application. It may be used at any time during program development, or may be integrated into an OEM system to aid in the discovery of latent errors. The Dynamic Debugger can be used to search for errors in any task, even while the other tasks in the system are running. The iRMX 86 Dynamic Debugger communicates with the developer via a terminal handler that supports full line editing.

System Crash/Dump Analyzer

The often difficult job of debugging real-time applications is made much simpler with the System Crash/Dump Analyzer. The analyzer allows program developers to record system memory for later analysis even if the system has halted. This analysis lists such vital information as which jobs have active tasks, which system queues contain which tasks, and what segments contain which data.

PARAMETER VALIDATION

Some iRMX 86 System Calls require parameters that may change during the course of developing iRMX 86 applications. The iRMX 86 Operating System includes an optional set of routines to validate these parameters to ensure that correct numeric values are used and that correct object types are used where the System expects to manipulate an object. For systems based only on the iRMX 86 Nucleus, these routines may be removed to improve the performance and code size of the System once the development phase is completed.

START-UP SYSTEMS

Two ready-to-run, multi-user start-up systems are included in the iRMX 86 Operating System package. These iRMX 86 start-up systems are fully configured, multi-user iRMX 86 Operating Systems ready to be loaded into memory by the Bootstrap Loader. Both start-up systems are configured to include all of the system calls for each layer and most of the features provided by iRMX 86. iRMX start-up systems include UDI support so that users may run languages such as PL/M-86, Pascal, FORTRAN, and software packages from independent vendors.

The start-up system for the iAPX 86 processor is configured for Intel SYSTEM 86/300 Series microcomputers with a minimum of 384K bytes of RAM. The following devices are supported.

- iSBC 215/iSBX 218 or iSBC 215G/iSBX 218A
- iSBC 254(S)
- Line Printer
- 8251A Terminal Driver
- iSBC 544 Terminal Driver

The start-up system for the iAPX 286 processor is configured for Intel SYSTEM 286/300 Series microcomputers with a minimum of 512K bytes and a maximum of 896K bytes of RAM. The following devices are supported.

- iSBC 208
- iSBC 215/iSBX 218 or iSBC 215G/iSBX 218A
- iSBC 254(S)
- Line Printer for iSBC 286/10
- 8274 Terminal Driver
- iSBC 544 Terminal Driver

Either system will run without hardware or software configuration changes and can be reconfigured on a standard system with at least 512K bytes of RAM. Definition files are also included for iSBC 186/03, 186/51 and 188/48 configurations.

This start-up system may be used to run the ICU (if a Winchester disk is attached to the system) to develop custom configurations such as those pictured in Figure 8. As shipped, the Human Interface supports a single user terminal. However, the Start-up System terminal configuration file may be altered easily to support from two to five users.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SPECIFICATIONS

Supported Software Products

iRMX 860	iRMX 86 Development Utilities Package, including the iAPX 86 and 88 Linker, Locater, Macro Assembler, Librarian, and the iRMX 86 Editor.
iRMX 861	PASCAL 86/88 Compiler
iRMX 862	FORTRAN 86/88 Compiler
iRMX 863	PL/M 86/88 Compiler
iRMX 864	TX Screen-oriented Editor
iMMX 800	MULTIBUS Message Exchange software package for iRMX 86, and 88 application systems
iOSP 86	Support Package for iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors
iRMX PSCOPE 86	High Level Language Debugger

Supported Hardware Products

COMPONENTS

iAPX 86 and 88 Microprocessors
iAPX 186 and 188 Microprocessors
iAPX 286 Microprocessors (Real Address Mode only)
8087 Numeric Data Processor Extension
80287 Numeric Data Processor Extension
iAPX 86/30 (80130) Operating System Firmware Component
8253 and 8254 Programmable Interval Timers
8259A Programmable Interrupt Controller
8251A USART Terminal Controller
8255 Programmable Parallel Interface
8274 Terminal Controller
82530 Serial Communications Controller

iSBC® MULTIBUS BOARD AND SYSTEM PRODUCTS

iSBC 86/12A, 86/05, 86/14, 86/30, 86/35, 88/25, and 88/40 Single Board Computers
iSBC 186/03 Single Board Computer
iSBC 186/51 Ethernet Controller
iSBC 188/48 Communications Controller
iSBC 286/10 Single Board Computer(Real Address Mode only)
iSBC 204 Diskette Controller
iSBC 206 Hard Disk Controller
iSBC 208 Diskette Controller
iSBC 215(G) Winchester Disk Controller
iSBX 218(A) Flexible Diskette Multi-Module Controller
iSBC 220 SMD Disk Hard Controller
iSBC 254(S) Bubble Memory System
iSBC 534 4-Channel Terminal Interface
iSBC 544 Intelligent 4-Channel Terminal Interface and Controller
iSBX 251 Bubble Memory Multi-Module
iSBX 350 Parallel Port (Centronics-type Printer Interface)
iSBX 351 Serial Communications Port
iSBX 270 CRT Light Pen and Keyboard Interface
SYSTEM 86/300 Family
SYSTEM 286/300 Family

AVAILABLE LITERATURE

The iRMX 86 Documentation Set is comprised of the following four volumes of reference manuals. Order numbers are associated with these four volumes only.

iRMX 86 INTRODUCTION AND OPERATOR'S REFERENCE MANUAL FOR RELEASE 6
Order Number: 146545-001

Introduction to the iRMX 86 Operating System
iRMX 86 Operator's Manual
iRMX 86 Disk Verification Utility Reference Manual

iRMX 86 PROGRAMMERS REFERENCE MANUAL FOR RELEASE 6, PART I
Order Number: 146546-001

iRMX 86 Nucleus Reference Manual
iRMX 86 Basic I/O System Reference Manual
iRMX 86 Extended I/O System Reference Manual

iRMX 86 PROGRAMMERS'S REFERENCE MANUAL FOR RELEASE 6, PART II
Order Number: 146547-001

iRMX 86 Application Loader Reference Manual
iRMX 86 Human Interface Reference Manual
iRMX 86 Universal Development Interface Reference Manual
Guide to Writing Device Drivers for iRMX 86 and iRMX 88 I/O Systems
iRMX 86 Programming Techniques
iRMX 86 Terminal Handler Reference Manual
iRMX 86 Debugger Reference Manual
iRMX 86 System Debugger Reference Manual
iRMX 86 Crash Analyzer Reference Manual
iRMX 86 Bootstrap Loader Reference Manual

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

iRMX 86 INSTALLATION AND CONFIGURATION GUIDE FOR RELEASE 6
Order Number: 146548-001

iRMX 86 Installation Guide

iRMX 86 Configuration Guide

Master Index for Release 6 of the iRMX 86 Operating System

Application Notes

Ap Note 130 — Using Operating System Processors to Simplify Microcomputer Designs. (Order Number: 230786-001

Ap Note 174 — Optimizing the iRMX 86 Operating System Performance on System 86/310 and System 86/330 (Order Number: 230990-001)

Training Courses

The iRMX 86 Operating System

Customer Seminars

Contact local Intel Sales Office for details on available video-tape and slide presentations.

ORDERING INFORMATION

The iRMX 86 Operating System is available under a number of different licensing options as noted here. Source listings are available on microfiche. Reconfigurable object libraries are provided on double density ISIS-formatted diskettes or on either double density, single sided iRMX 86-formatted 8″ diskettes, or double density, double sided, 5.25″ diskettes. ISIS-format diskettes may be used on Intel Intellec Development Systems. The iRMX 86-format may be used on any iRMX 86-based system supporting the appropriate compilers and development environment.

The OEM license options listed here allow users to incorporate the iRMX 86 Operating System into their applications. Each use requires payment of an Incorporation Fee.

ORDER CODE	DESCRIPTION
iRMX 86 KIT BRO:	Double density, single-sided 8″ ISIS format OEM license
iRMX 86 KIT ERO:	Double density, single sided 8″ iRMX 86-Format OEM license for use on iRMX 86-based environments.
iRMX 86 KIT JRO:	Double density, double sided 5.25″ iRMX 86-Format OEM license for use on iRMX 86-based environments.

Other licensing options include prepayment of all future incorporation fees, single use rights for a single machine, use at a second development site, one year update service extensions, the right to make copies for additional development systems, and source listing materials.

Each option includes 90 days of support service that provides the quarterly iRMX 86 Technical Report, Software Problem Report Service, and copies of System Updates that occur during this period. Except for source listings, all initial licenses include a complete set of iRMX 86 Documentation.

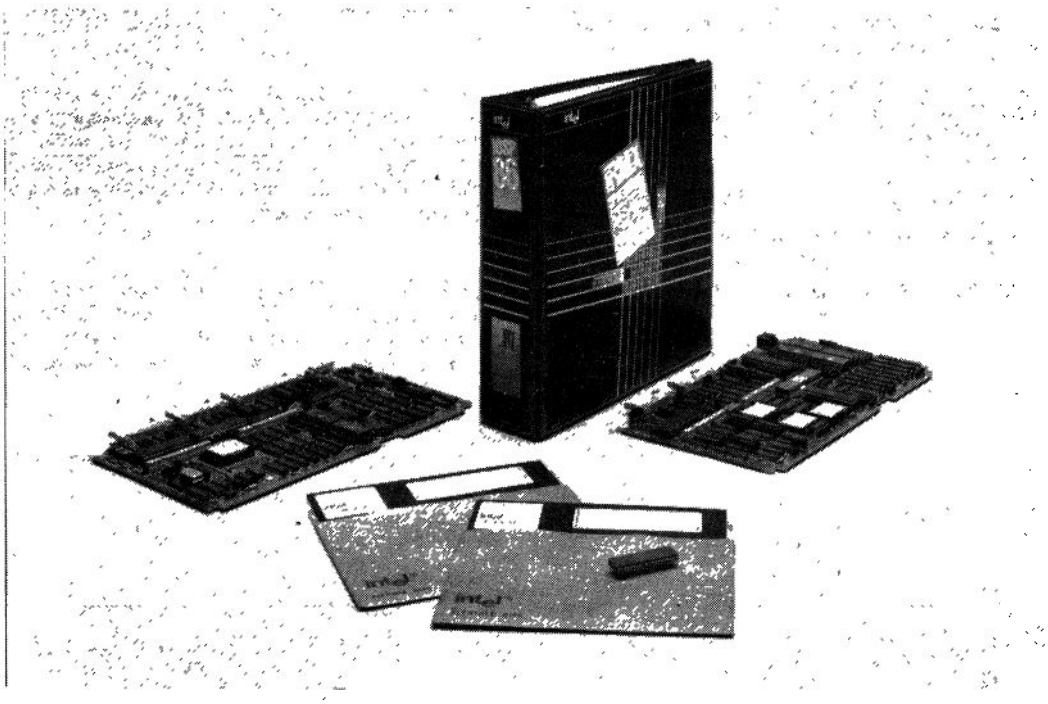
As with all Intel software, purchase of any of these options requires the execution of a standard Intel Master Software License. The specific rights granted to users depends on the specific option and the License signed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

iOSP™ 86
iAPX 86/30, iAPX 88/30, iAPX 186/30 and iAPX 188/30 SUPPORT PACKAGE

- **Development and run-time support for iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors**
- **Total iRMX™ 86 Operating System software compatibility**
- **Extendable with iRMX™ 86 Operating System calls**
- **Compatible with Intel® PL/M 86, PASCAL 86, FORTRAN 86, and ASM86 MACRO ASSEMBLER**
- **Supports (P)ROM or RAM based system**
- **Supports custom system initialization**
- **Interactive Configuration Utility**

The Intel iOSP™ 86 Support Package for the iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors contains a comprehensive set of easy-to-use tools needed to develop (P)ROM or RAM-based applications that use the 80130 Operating System Firmware component. This Support Package is compatible with all versions of the 80130 component. All of the system initialization and run-time facilities are provided in libraries that may be configured to specific requirements, and linked to application programs written in either ASM86 MACRO ASSEMBLER or a high level programming language such as PASCAL 86, FORTRAN 86, and PL/M 86. The iOSP 86 Package provides users with the basic initialization and interface routines needed to build application software based on the fundamental operating system functions of the iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors. The iOSP 86 Package also enables users to add higher level I/O functions from the fully compatible iRMX™ 86 Operating System, or to form custom, real-time systems.



Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No other Circuit Patent Licenses are implied.

©INTEL CORPORATION, 1984

JUNE, 1984
ORDER NUMBER: 210236-002

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FUNCTIONAL DESCRIPTION

The iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors (OSPs) provide an easy-to-use foundation on which many real-time applications may be built. They provide the functions and system support needed to implement both simple and complex applications that require multiple tasks to run concurrently (see Figure 1). These services are made possible by the addition of the five new data types integrated into the 80130 Operating System Firmware (OSF) component. The 80130 OSF extends the basic data types of the CPU (integer, byte, character, etc.) by adding new system data types (JOB, TASK, MAILBOX, SEGMENT, and REGION), and extensive timer, interrupt, memory, and error management designed to give real-time response to multitasking and multiprogramming applications. As shown in the second half of the figure, other operating system functions such as mass storage I/O services and an easy-to-use Human Interface can be added easily, by using modules from the iRMX 86 Operating System. The iOSP 86 Support Package provides both an interface between application software and the Operating System Processors, and development tools designed to make the implementation and initialization of real-time, multitasking systems much easier.

The iOSP 86 Support package provides system developers with the configuration options necessary to tailor the iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors to custom applications. Central to the entire configuration process is the Interactive Configuration Utility (ICU86). This utility is an easy-to-use tool which allows you to make configuration decisions by responding to screen-oriented displays. Using the ICU, users can build the necessary support code. The interface libraries form a simple interface between application software and the operating system primitives of the 80130 OSF component.

Memory and I/O Addressing

The 80130 OSF requires that a 16K byte block of memory address space be reserved for accessing internal functions. The ICU is used to specify the base address of the 80130 and the beginning of the initialization support code.

All Interrupt and Timer management of the OSF is controlled via a reserved 16 byte I/O address block that may be selected by the user. In addition, from 1 to 7 slave 8259A interrupt controllers can be specified in order to provide the system with up to 57 priority interrupt sources. The 80130 baud rate generator may also be configured to support an optional terminal interface.

Extending the 80130 OSF

The 80130 OSF allows users to add their own operating system extensions. These extensions may take advantage of the detailed and efficient intertask communication and synchronization primitives already provided by the 80130, and/or may utilize custom functions tailored to specific applications. The Support Package also enables users to extend the OSF with the extensive services of Intel's iRMX 86 Operating System, thereby allowing applications to grow without having to change or alter application software already written, or having to write other operating system software.

Use of the 80130 OSF with the iRMX 86 Operating System reduces the amount of memory needed for the iRMX 86 Nucleus layer by 14K bytes, and enables applications to take advantage of the increased

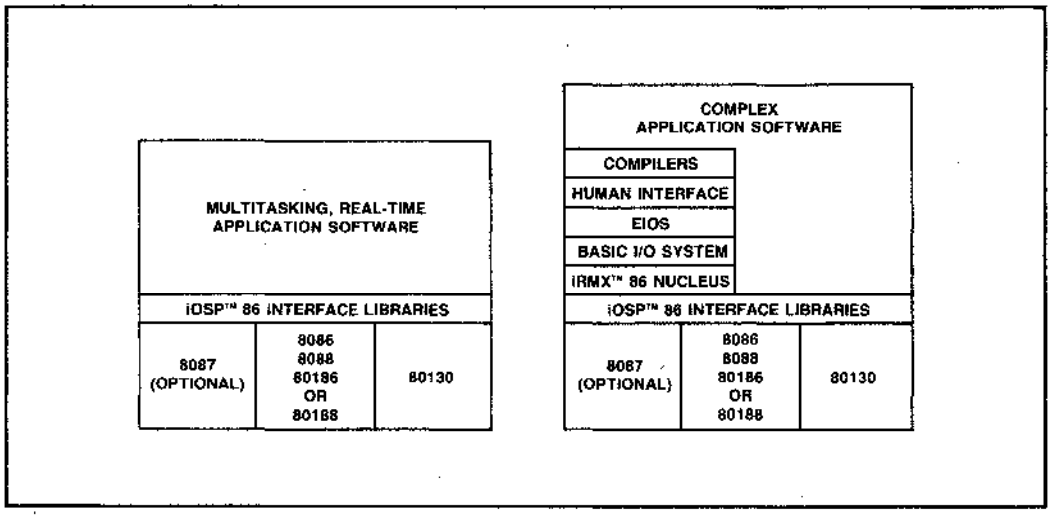


Figure 1. Structure of Typical Systems

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

performance and reduced size requirements inherent in the iAPX 86/30, 88/30, 186/30, and 188/30 Operating System Processors. Since each of the services provided by the 80130 component is totally compatible with iRMX 86, applications have an automatic upward path to support complete file systems and multiple processor environments.

Application Interfaces

Two interface libraries are included in the iOSP 86 Support Package. The first allows programmers to write application software modules in the Compact Model of computation supported by Intel's compilers. The second provides an interface to program segments written in either the Medium or Large Models. The iOSP 86 Support Package does not support program segments written in the Small Model.

The interface libraries provide the means of accessing all of the primitives supported by the Operating System Processors. With this interface, and all the memory management primitives of the OSPs, applications have full access to 1M byte of memory, and all of the addressing modes of the CPU.

These libraries are fully compatible with object modules produced by the ASM86 MACRO ASSEMBLER, and the PASCAL 86, FORTRAN 86, and PL/M 86 Compilers.

Application Initialization

The iOSP 86 Support Package provides, via the ICU, for the configuration of the system ROOT JOB, and all user application JOBs that require initialization when the system is started. The user also specifies the configuration of the interrupt system (including the optional iAPX 186/188 interrupt controller in either master or slave modes and any slave 8259A interrupt controllers) and the clock rate used for system timing. These choices are automatically programmed into the various devices when the system is initialized.

Parameter Validation

Parameter validation is a configuration option of an OSP-based system. The OSP can check the parameters of the primitive that you invoke either on a systemwide basis or on a per job basis.

Operating System Calls

The 80130 OSF performs a total of 38 operating system primitives all of which are completely compatible with the equivalent iRMX 86 Operating System calls. The iOSP 86 Support Package provides user-level interfaces to these primitives to enable applications to create, delete, control, and exchange the new data types provided by the 80130 OSF. In general, these interfaces allow application software to manage all of the resources of an iAPX 86/30, 88/30, 186/30, or 188/30 OSP (and an optional 8087 Numeric Processor Extension) system via any of the 38 system calls shown in Figure 2.

Required Development Hardware

Use of the iOSP 86 Support Package requires a Series III Intellec Development System with double density flexible diskette drives or any iRMX 86 system supporting a standard 5.25 inch or 8 inch flexible diskette drive and the iRMX 860 Assembler and Utilities Package. Use of the 80130 requires only a minimal system including either the iAPX 86/30, 88/30, 186/30 or 188/30 Operating System Processor, and enough system memory to contain the application programs and initialization and interface software provided in the iOSP 86 Package.

Board Level Product Support

Intel microcomputer boards which use the 80130 OSF include the iSBC 186/03 and the iSBC 186/51 Single Board Computers. An iOSP 86 application may be written specifically to run on these boards.

JOB GROUP	SEGMENT GROUP	INTERRUPT MANAGEMENT GROUP
CREATE JOB	CREATE SEGMENT	SET OS EXTENSION
END INIT TASK	DELETE SEGMENT	SET INTERRUPT
		ENTER INTERRUPT
TASK GROUP	REGION GROUP	EXIT INTERRUPT
CREATE TASK	CREATE REGION	WAIT INTERRUPT
DELETE TASK	DELETE REGION	SIGNAL INTERRUPT
SUSPEND TASK	SEND CONTROL	RESET INTERRUPT
RESUME TASK	RECEIVE CONTROL	ENABLE
SLEEP	ACCEPT CONTROL	DISABLE
GET TASK TOKENS		GET LEVEL
SET PRIORITY	OBJECT MANAGEMENT GROUP	
	CATALOG OBJECT	ERROR CONTROL GROUP
MAILBOX GROUP	LOOKUP OBJECT	SET EXCEPTION
CREATE MAILBOX	DISABLE DELETION	SIGNAL EXCEPTION
DELETE MAILBOX	ENABLE DELETION	GET EXCEPTION
SEND MESSAGE	GET TYPE	
RECEIVE MESSAGE		

Figure 2. Operating System Primitives

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Part Number	Description
OSP 86 B	iOSP 86 Support Package contained on an ISIS-II compatible, single-sided, double density 8 inch diskette.
OSP 86 E	iOSP 86 Support Package contained on an iRMX 86 format, single-sided, double density 8 inch diskette.
OSP 86 J	iOSP 86 Support Package contained on an iRMX 86 format double-sided double density, 5.25 inch, 48 tracks-per-inch diskette.

ORDERING INFORMATION

Each of the ordering options listed below include all the necessary initialization and interface procedures needed to use the iAPX 86/30, 88/30, 186/30, and 188/30 Operating System processors. Purchase of the iOSP 86 Package requires verification of an Intel Master Software License. Each package also includes an iOSP 86 User's Manual (Document Number 146798-001), and a 90 day update service.

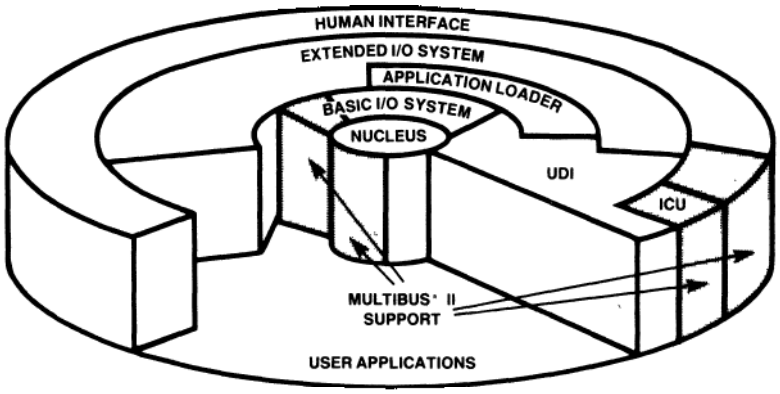
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PRELIMINARY

iRMX™ 86-MULTIBUS® II SUPPORT PACKAGE

- **MULTIBUS® II support for iSBC® 286/100 applications in Real Address Mode, including support for the SCSI peripheral interface and up to 1 megabyte addressability**
- **Functions in conjunction with the iRMX™ 86 Release 6 Operating System**
- **Interprocessor Signal Support**
- **Automatic software configuration of memory boards**
- **Support for battery backed-up, global time-of-day clock**
- **Extendable to allow addition of custom device drivers**

The iRMX™ 86-MULTIBUS® II Support Package, functioning with the iRMX 86 Release 6 Operating System software, provides the ability to execute all configurable layers of the iRMX 86 software in the MULTIBUS II environment (iRMX 86-MULTIBUS II Operating System). Applications in Real Address Mode are supported for the iSBC® 286/100 board, including support for the SCSI peripheral interface and all iSBX™ boards supported by iRMX 86 Release 6, as well as support for iAPX 286 component applications.



NEW IN IRMX™ 86 MULTIBUS II OPERATING SYSTEM

iRMX™ VLSI Operating System

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

© INTEL CORPORATION, 1984

SEPTEMBER, 1984
ORDER NUMBER: 280057-001

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FUNCTIONAL DESCRIPTION

Overview

The iRMX 86 MULTIBUS II package contains system modules that replace portions of the iRMX 86 Release 6 Operating System, allowing the iRMX 86 Operating System to execute in a MULTIBUS II environment. All the functions available in the IRMX 86 Operating System are available in the iRMX 86-MULTIBUS II Operating System. For a complete description of these functions, their value, and performance, please refer to the Release 6 iRMX 86 Operating System Data Sheet (order number 210885-002).

This functional description section describes the new features provided by the iRMX 86 MULTIBUS II package. These new features add the new capabilities required for OEMs to execute the iRMX 86 Operating System in a MULTIBUS II environment for iSBC 286/100 or iAPX 286 applications in Real Address Mode.

Interprocessor Signal Support

In a MULTIBUS II system, interprocessor communication and synchronization is done via messages over the bus. This communication includes data-less messages to signal that an event has occured. The iRMX 86 MULTIBUS II package supports signal messages using the Message Interrupt Controller (MIC) Component. The major advantage of signal message support is the ability for a host cpu board to send or receive signal messages from up to 254 distinct sources, with the priorities of each message being based on the sending or receiving task's priority. Signal messages are not tied to hardware interrupt levels and priorities as external interrupts were in the MULTIBUS I environment.

Automatic Software Configuration of Memory Boards

The iRMX 86-MULTIBUS II Operating System has the option of automatically configuring memory boards. The addresses for each board are defined sequentially in relation to the physical placement of each board in the card cage. This feature allows for the swapping, adding, and deleting of memory boards in the system on a dynamic basis.

Accurate Time-of-Day Clock Support

Resident in every MULTIBUS II system is a Central Services Module (iSBC CSM/001 board). The CSM board contains a battery backed-up, global time-of-day clock. The iRMX 86-MULTIBUS II Operating System uses this clock to automatically initialize the time-of-day clock maintained by the operating system.

Custom Device Driver Support

Like the iRMX 86 Operating System, the iRMX 86-MULTIBUS II Operating System is extendable to support user value-added custom device drivers. This feature allows the system to be more closely tailored to meet a specific application requirement and expands the list of supported hardware products. The user need not purchase source code to write a custom driver and can configure the driver into the system at configuration time. Custom drivers can use the Message Interrupt Controller (MIC) to pass signal messages.

SPECIFICATIONS

Below is the list of supported products for the iRMX 86 MULTIBUS II Support Package.

Supported Software Products

iRMX 86 Release 6 Operating System

Supported Hardware Products

Components:

iAPX 286 Microprocessor (Real Address Mode only)
80287 Numeric Data Processor Extension
8253 and 8254 Programmable Interval Timers
8259A Programmable Interrupt Controller (PIC)
8255 Programmable Parallel Interface (PPI)
82530 Serial Communications Controller (SCC)
82258 Advanced DMA Controller (ADMA)
Bus Arbiter Controller (BAC)
Message Interrupt Controller (MIC)

iSBC® MULTIBUS® II Board Products:

iSBC 286/100 Single Board Computer (Real Address Mode only)
iSBC CSM/001 Central Services Module
iSBC MEM/312, 310, 320, 340 cache-based memory
iSBX 218(A) Flexible Diskette Multi-Module Controller
iSBX 251 Bubble Memory Multi-Module
iSBX 270 CRT Light Pen and Keyboard Interface
iSBX 350 Parallel Port (Centronics-type Printer Interface)
iSBX 351 Serial Communications Port

AVAILABLE LITERATURE

iRMX 86-MULTIBUS II Support Package Reference Manual (order number 147127)

There are four manual kits supplied with the iRMX 86 Release 6 Operating System and are available under the order numbers shown in the iRMX 86 Operating System Data Sheet (order number 210885-002)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ORDERING INFORMATION

The iRMX 86 MULTIBUS II Package is available under a number of different licensing options. Obtaining a license for the iRMX 86 Release 6 Operating System is a pre-requisite to licensing the iRMX 86 MULTIBUS II Package. Reconfigurable object libraries are provided on: 1) Double-density single-sided ISIS-formatted 8″ diskettes; 2) Double-density, single-sided iRMX 86-formatted 8″ diskettes; 3) Double-density, double-sided, iRMX 86-formatted 5.25″ diskettes. ISIS-format diskettes may be used on Series III Development Systems. The iRMX 86-format may be used on Series IV Development Systems (5.25″ diskettes) or any iRMX 86-based system supporting the appropriate disk drivers, compilers and development environment.

The OEM license options listed here allow users to incorporate the iRMX 86 MULTIBUS II package into their applications. Each use requires payment of an Incorporation Fee.

ORDER CODE	DESCRIPTION
iRMX 86 II BRO:	Double-density, single-sided 8″ ISIS-format OEM license.
iRMX 86 II ERO:	Double-density, single-sided 8″ iRMX 86-format OEM license.
iRMX 86 II JRO:	Double-density, double-sided 5.25″ iRMX 86-format OEM license.
iRMX 86 II KIT BRO:	Includes iRMX 86 Release 6. Double-density, single-sided 8″ ISIS format OEM license.
iRMX 86 II KIT ERO:	Includes iRMX 86 Release 6. Double-density, single-sided 8″ iRMX 86-format OEM license.
iRMX 86 II KIT JRO:	Includes iRMX 86 Release 6. Double-density, double-sided 5.25″ iRMX 86-format OEM license.

Other licensing options include prepayment of all future incorporation fees and single use rights for a single machine.

Each option includes 90 days of support service that provides Software Problem Report Service and copies of System Updates that occur during this period.

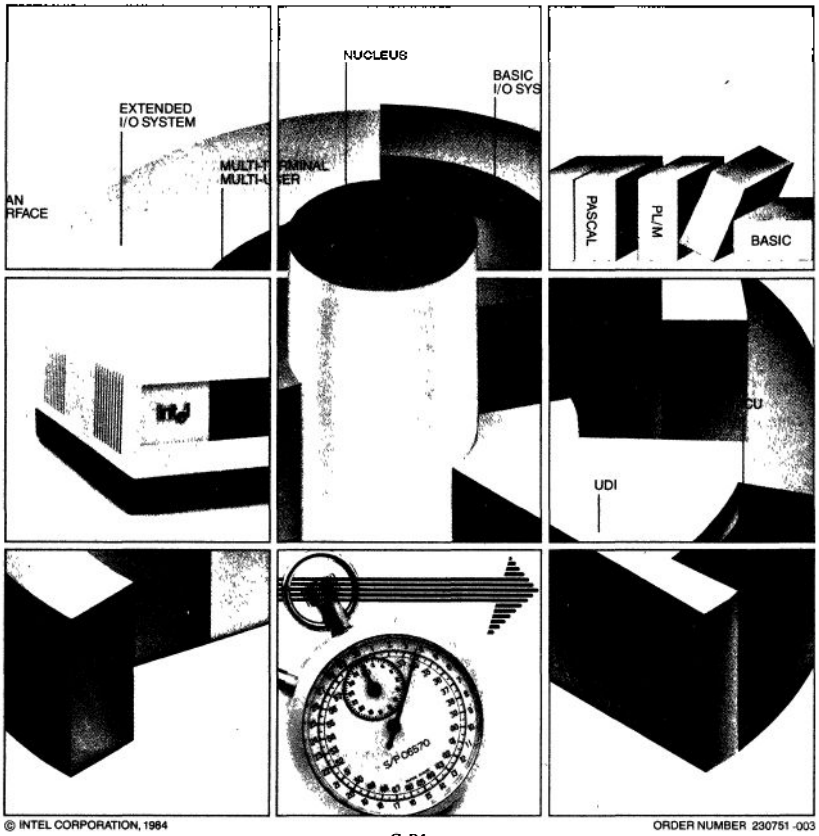
As with all Intel software, purchase of any of these options requires the execution of a standard Intel Master Software License. The specific rights granted to users depend on the specific option and the license signed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

iRMX™ OPERATING SYSTEM

- **High-performance, real-time, multi-tasking operating system for Intel's 86/300 and 286/300 microcomputer systems**
- **Highly configurable, modular structure for easy system expansion**
- **Wealth of design facilities and industry-standard languages to support fast, easy development**
- **Application software portable to next generation of Intel VLSI**
- **Supported by Intel's post-sales software support organization**



© INTEL CORPORATION, 1984

ORDER NUMBER 230751-003

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Total Solution for the Real-Time Application OEM

Intel's iRMX™ 86 Operating System is a real-time, multi-tasking, multiuser, multiprogramming operating system designed to support high performance, time-critical applications such as factory automation, industrial control and communications networks. The iRMX operating system serves as an optimized event-driven executive for managing and extending the resources of Intel's 86/300 and 286/300 systems in real-time applications where high speed and low interrupt latency are required. Added performance for demanding numeric-intensive tasks comes from support of Intel's floating point math coprocessors.

Comprised of modular layers, Intel's iRMX operating system is highly configurable, allowing the OEM to easily customize the system to meet the needs of target applications. In addition, the iRMX operating system provides OEMs with complete development capabilities. It has systems debuggers, crash analyzers, screen editors, utilities, and an Interactive Configuration Utility (ICU)—everything the development engineer needs to design and configure efficiently.

To further reduce development time, a complete set of industry-standard languages enables OEMs to take advantage of existing application software. This shaves months off development time and is a key advantage to the competitive OEM.

Speed, the Name of the Real-Time Game

In a real-time system the computer must respond to interrupts instantly; time is always at a premium. Intel's iRMX Operating System delivers superior real-time performance, thanks to ultra-fast context switching, task synchronization and memory-based message passing.

The iRMX 86 Operating System manages the resources of the 286/300 systems in real-address mode. iRMX 86 makes possible the utilization of the high-



performance capabilities of Intel's iAPX 286 microprocessor for those demanding high-speed applications.

Further accelerating processing power in number-crunching and floating point math applications is iRMX operating system's support of Intel's math coprocessors.

Our 8087 numeric data processor in our iRMX 86-based systems can perform floating point operations four times faster than competitive minicomputers with hardware math processors. For even greater performance, OEMs can select the iAPX 286 and the 80287 coprocessor working in tandem in the iRMX 86 system.

The superior price/performance ratio that results from combining Intel's iRMX operating systems and the System 300 family makes the choice clear: a more competitive Intel micro-based system over a more expensive minicomputer-based system.

Add More Processors for More Power, More Speed

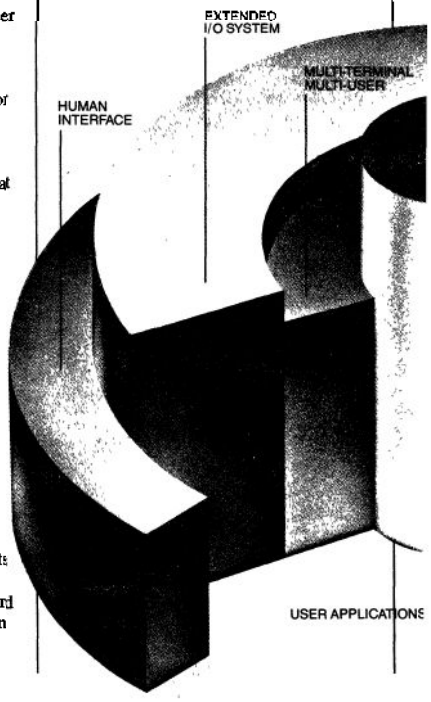
Need still more micro-muscle in your application? In an iRMX-based system, additional intelligent boards can be added to enhance system throughput.

With the iMMX™ 800 (MULTIBUS® Message Exchange) software package, the iRMX 86 Operating System supports a loosely-coupled multiprocessing environment. Tasks running on one board may communicate with tasks running on other boards, even if they operate under different Intel operating systems or microprocessors.

Multiprocessing is possible due to the hardware capabilities of Intel's System 300 MULTIBUS System Bus and the software support provided by iMMX™ 800. Overall system performance and flexibility can be greatly enhanced by off-loading the main CPU with such intelligent I/O boards as Intel's quad serial communication controller, digital controller or Ethernet communications controller.

Modular Software for Versatile, Easy Configuration

The iRMX operating systems shipped with Intel's 86/300 and 286/300 hardware systems are preconfigured at the factory to support a standard board set; however, the OEM can additionally configure or

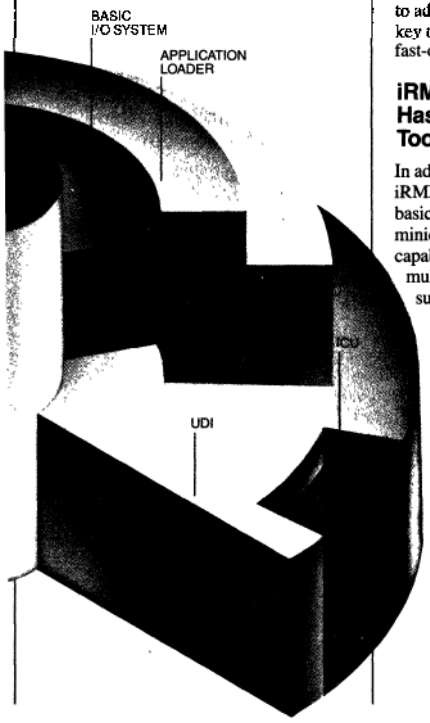


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

extend the operating system to meet specific needs.



Intel's iRMX operating systems are configurable by system layer and by system call within each layer. Such flexibility gives designers the ability to choose software features that best suit their application's size and functional requirements. The iRMX Operating System also includes I/O drivers for many of Intel's MULTIBUS boards and industry-standard peripherals. You simply select the ones you need.



The Interactive Configuration Utility (ICU) is a built-in facility for assisting the OEM in the configuration process. The ICU prompts the user for system parameters and requirements, then builds a command file to compile, assemble, link, and locate necessary files.

The net results for the OEM: fast, easy system configuration with quick time-to-market benefits.

For customizing and extending your iRMX system, Intel has provided all the "hooks" necessary to make the job easy. The iRMX 86 Operating System contains extendability features that enable the OEM to add custom operating system calls, custom features, and custom functionality to his application—at any time in the application's life. The ability to add functions late in a product's life is key to an OEM's competitive edge in a fast-changing market.

iRMX™ Operating System Has All the Fundamentals, Too!

In addition to multiprocessing, Intel's iRMX operating systems have all the basics you would expect to find in a minicomputer operating system . . . capabilities such as multitasking, multiprogramming, and multiterminal support.

Multitasking requires a method of managing the different processes of an application and for allowing these processes to communicate with each other. The iRMX Nucleus provides these facilities plus task scheduling. The Basic I/O System provides users with the system calls for direct management of I/O devices needed for real-time applications. The Extended I/O System adds a number of I/O management capabilities to simplify access to files, such as automatic buffering and synchronization of I/O requests.

The Human Interface functions give users and applications simple access to the file and system management capabilities. Using the multiterminal support provided by the Basic I/O system, the Human Interface can support several simultaneous users. For example, multi-terminal support allows one person to use the iRMX Editor, while another compiles a FORTRAN or Pascal program, while several others load and access applications.

On-Target Development: One System Does It All

The beauty of Intel systems lies in their flexibility. Engineers developing an iRMX-based target system can use the same iRMX-based system in the development process; the development and target systems are one in the same. The bottom-line benefit is low entry-level costs for the OEM.

On-target development contributes immeasurably to a shorter development curve and decreased time-to-market, since it isn't necessary to purchase and learn separate development systems. With Intel's iRMX-based system, one system does it all.



Tap into a Wide Range of Languages and Utilities

An Intel iRMX-based system supports many industry-standard and widely available languages: FORTRAN 77, Pascal (ISO Draft Standard) and PL/M compilers; Intel Assemblers, and popular independent vendor products, such as Microsoft's BASIC and Mark Williams' C compiler.

iRMX operating systems also have a menu-driven, screen-oriented text editor and a variety of utilities for manipulating

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

object code to facilitate the development process.

Multiple-language support is made possible by a set of systems calls known



as the Universal Development Interface (UDI) which enables the iRMX systems to interface with many compilers and language translators. UDI ensures that users will be able to transport applications to future releases of iRMX operating systems as well as use language and utilities of other software vendors that support UDI. (For more information on Intel iRMX languages, see the iRMX Language Fact Sheet)

As an option, a commercial extension package iCEX is available. It provides such useful utilities as: a Shared I/O System (SIOS) that allows multiple tasks to access mass storage data through shared buffers in main storage; a Re-entrant Program Manager (RPM) that eliminates the need to have multiple copies of the same program in memory to support concurrent applications; a File Printer; Multi-user LOG ON facilities; and many more.

Intel's Open Systems Approach Means Freedom to Grow

At Intel, we believe that systems need to expand in order to meet the needs of a changing market; and that is how we design our products.

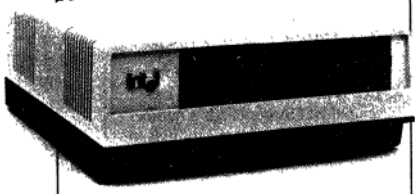
Standards are the key to systems that are open to future expansion, future technology and future markets.

Intel's iRMX operating systems are built from the inside-out with industry standards: UDI (Universal Development Interface), RTI (Runtime Interface), MULTIBUS System Bus (IEEE 796), iMMX 800 Package (MULTIBUS multi processing), Ethernet (IEEE 802.3), extended math format (IEEE P754), and industry-standard peripheral device interfaces.

An OEM who builds his product around one of Intel's RMX-board systems is assured of multi-vendor hardware/software alternatives and a future upgrade path. In today's highly competitive markets, that is the only kind of system to build.

Today, you'll have the ability to tap into readily available application software packages, languages, and utilities, MULTIBUS boards, and peripherals. Tomorrow, you will be able to tap into the latest, high-performance VLSI without sacrificing today's software investment. Applications written on iRMX 86 will run on Intel's iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286-based systems.

Not to be forgotten are the advantages of starting from the systems level to begin with. Intel has invested hundreds of man-years in software and hardware development for its systems products. For the OEM trying to meet a market window, time-to-market is much faster when starting with a system instead of boards or components. It makes good business sense to let Intel provide the "micro-engine", so you can concentrate on your area of expertise and get to market sooner!



Worldwide Service and Support

The iRMX 86 Operating System is a mature proven product with thousands of installations at the component, board and systems levels. Post-sales software support is available to Intel iRMX 86 Operating System OEMs in the form of software updates and routine systems software maintenance. Software support is extendable in one-year increments after the initial 90-day warranty. Hotline service is available separately to customers needing quick regional software support. All software is completely documented, and users receive monthly technical reports, newsletters and access to the iRMX users group and software libraries.

iRMX users can also take advantage of Intel's worldwide staff of trained hardware and software engineers for application design assistance. We offer complete training for operating system software and associated system hardware, bringing OEM's up to speed and helping get their products to market quickly.

Intel, the Technology Leader ...With the Total Solution

Intel started the microprocessor revolution with the 4004 and has been the market leader with every generation of advanced microprocessor VLSI since. We not only invented the microprocessor but MULTIBUS single board computers, as well.

Intel's technology leadership has, by necessity, extended from microprocessors into operating system software. iRMX is recognized as the industry standard real-time VLSI operating system.

OEMs can enhance their product's marketability by leveraging their value-added on top of the solid foundation of an iRMX-based Intel 300 microcomputer system. Intel's solution offers the most price/performance with the least risk to progressive OEMs . . . because we know the real-time game from the inside out.

RMX™ OPERATING SYSTEM

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Specifications

Supported Software Products

iRMX 860	iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, Macro Assembler, Librarian, and the iRMX 86 Editor
iRMX 861	Pascal 86/88 Compiler
iRMX 862	FORTRAN 86/88 Compiler
iRMX 863	PL/M 86/88 Compiler
iRMX 864	TX-Screen-Oriented Editor
iRMX 865	BASIC Interpreter
iRMX 866	C Compiler
iMMX 800	MULTIBUS® Message Exchange software package for iRMX 80, 86, 88, and 286 application systems

Supported Hardware Products

iSBC® MULTIBUS® Products

iSBC 86/12A, 86/05, 86/14, 86/30, 86/35, 88/25, 88/40, and 286/10	Single Board Computers
iSBC 186/03	Single Board Computer
iSBC 186/51	Ethernet Controller
iSBC 188/48	Communications Controller
iSBC 286/10	Single Board Computer (Real Address Mode only)
iSBC 204	Flexible Disk Controller
iSBC 206	Hard Disk Controller
iSBC 208	Flexible Disk Controller
iSBC 215	Winchester Disk Controller
iSBC 220	SMD Disk Controller
iSBX 251	Bubble Memory System
iSBC 254	Bubble Memory System
iSBC 534	4-Channel Terminal Interface
iSBC 544	Intelligent 4-Channel Terminal Interface and Controller
iSBX 218	Flexible Disk Controller
iSBX 350	Parallel Port (Centronix-type Printer Interface)
iSBX 351	Serial Communications Port
iSBX 270	CRT, Light Pen and Keyboard Interface

System 86/300 Family
System 286/300 Family

Available Literature

The iRMX 86 Documentation Set is comprised of the following four volumes of reference manuals. Order numbers are associated with these four volumes only.

iRMX 86 Introduction and Operator's Reference Manual for Release 6
Order Number: 146545-001

Introduction to the iRMX 86 Operating System
iRMX 86 Operator's Manual
iRMX 86 Disk Verification Utility Reference Manual

iRMX 86 Programmers Reference Manual for Release 6, Part 1
Order Number: 146546-001

iRMX 86 Nucleus Reference Manual
iRMX 86 Basic I/O System Reference Manual
iRMX 86 Extended I/O System Reference Manual

iRMX 86 Programmer's Reference Manual for Release 6, Part II
Order Number 146547-001

iRMX 86 Application Loader Reference Manual
iRMX 86 Human Interface Reference Manual
iRMX 86 Universal Development Interface Reference Manual
Guide to Writing Device Drivers for iRMX 86 and iRMX 88 I/O Systems
iRMX 86 Programming Techniques
iRMX 86 Terminal Handler Reference Manual
iRMX 86 Debugger Reference Manual
iRMX 86 System Debugger Reference Manual
iRMX 86 Crash Analyzer Reference Manual
iRMX 86 Bootstrap Loader Reference Manual

iRMX 86 Installation and Configuration Guide for Release 6
Order Number 146548-001

iRMX 86 Installation Guide
iRMX 86 Configuration Guide
Master Index for Release 6 of the iRMX 86 Operating System

iRMX™ 86 Configuration Size Chart

System Layer	Min. ROMable Size	Max. Size	Data Size
Bootstrap Loader	1K	1.5K	6K*
Nucleus	10.5K	24K	2K
BIOS	26K	78K	1K
Application Loader	4K	10K	2K
EIOS	10.5K	12.5K	1K
Human Interface	22K	22K	15K
UDI	11K	11K	0
Terminal Handler	3K	3K	0.3K
Debugger	28.5K	28.5K	1K
Human Interface Commands			116K
Interactive Configuration Utility			308K

System 86/300 Memory:	348KB
Maximum Addressable Memory:	1MB
Minimum Memory Required with ICU Loaded:	448KB

*Usable by System after Bootloading

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

Ordering Information

Each iRMX operating system includes two startup systems supporting Intel's System 300 standard hardware and Intel processor boards. Intel System customers also receive the iRMX 860 (Assembler, Linker, Locator, Libraries, Editor, Utilities) and iRMX 863 (PL/M Language) products and are entitled to one prepaid incorporation fee. Also included: Software Problem Reporting Service (SPR), and a 90 day System Software Subscription (new s/w release updates). Also includes System Software documentation.

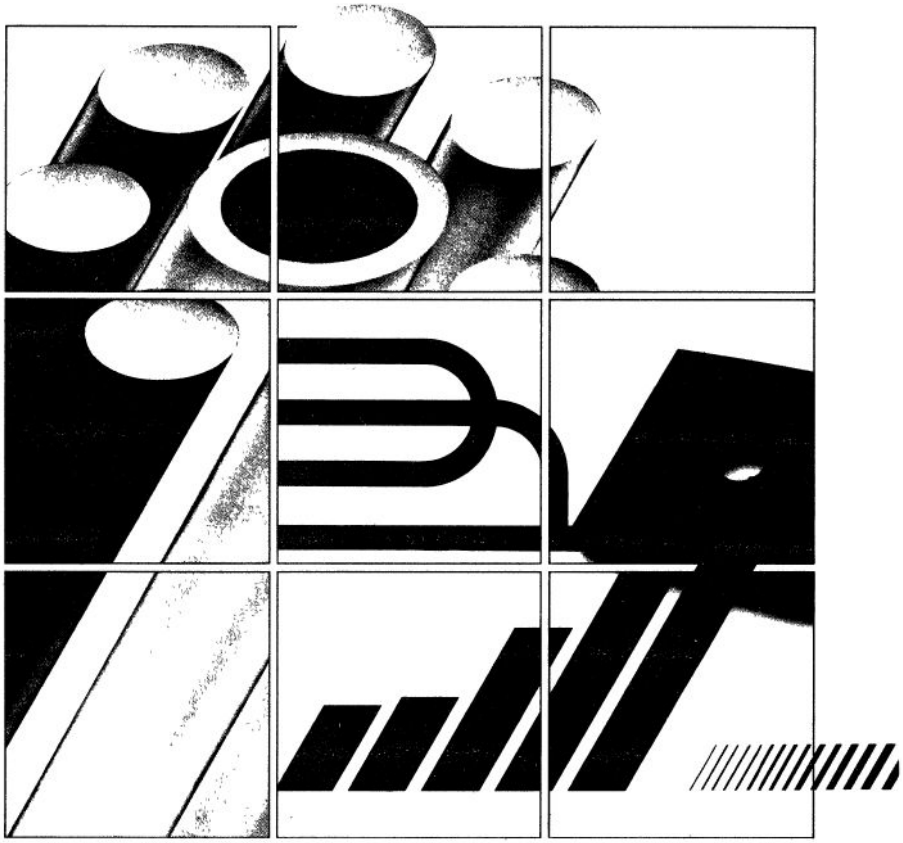
Refer to Intel's OEM price list, OEM Microcomputer System section, for ordering information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

XENIX* 3.0 OPERATING SYSTEM

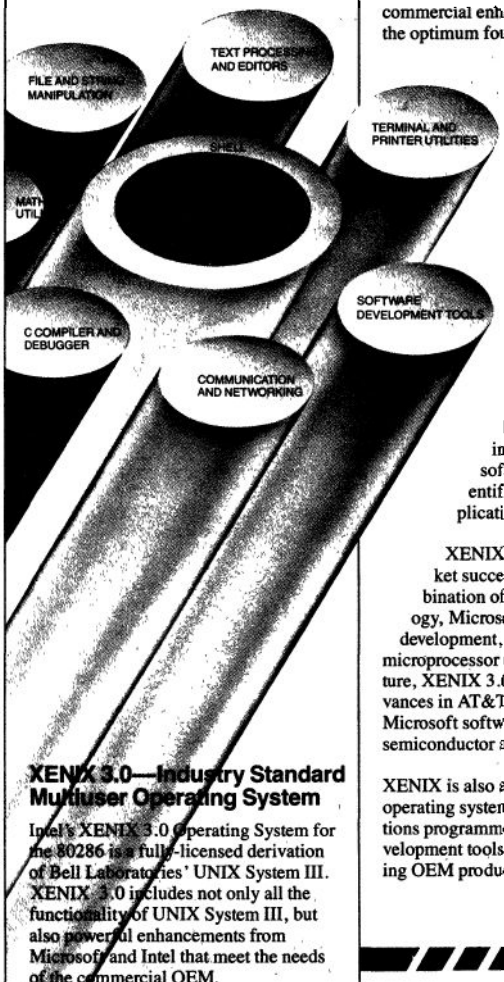
- **XENIX 3.0 Industry Standard Multiuser Operating System**
- **Fully licensed version of the UNIX† operating system optimized for the Intel 80286 processor**
- **Leading edge microprocessor implementation of UNIX, fastest floating point performance on a microprocessor**
- **Important commercial OEM enhancements**
- **Supports multiple levels of integration: components, boards and systems**
- **Supported by Intel's worldwide post-sales service and support organizations**



©INTEL CORPORATION 1984
*XENIX is a trademark of Microsoft Corporation
†UNIX is a trademark of Bell Laboratories

ORDER NUMBER 230752-003

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



XENIX 3.0—Industry Standard Multiuser Operating System

Intel's XENIX 3.0 Operating System for the 80286 is a full-licensed derivation of Bell Laboratories' UNIX System III. XENIX 3.0 includes not only all the functionality of UNIX System III, but also powerful enhancements from Microsoft and Intel that meet the needs of the commercial OEM.

The Best Foundation for Building OEM Solutions

XENIX 3.0 provides the OEM with a complete software base on which to build value-added functionality. It includes the operating system, the C language, text processors, development tools, system accounting and security features, and commercial enhancements that make it the optimum foundation for OEM application software solutions.

*XENIX is a trademark of Microsoft Corporation

XENIX: Portable, Flexible, Powerful

XENIX has become the industry-standard microcomputer operating system for interactive, multi-user applications. It has gained wide popularity in applications such as distributed data processing, business data processing, word processing, software development, scientific and engineering applications, and graphics.

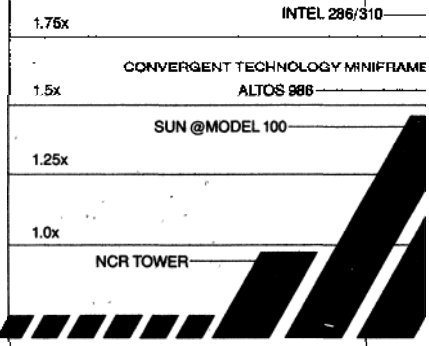
XENIX has achieved this market success through a solid combination of UNIX system technology, Microsoft value-added product development, and Intel's experience in microprocessor technology. In the future, XENIX 3.0 will benefit from advances in AT&T UNIX technology, Microsoft software technology and Intel semiconductor and system technology.

XENIX is also an extremely powerful operating system, providing the applications programmer with a wealth of development tools and utilities for bringing OEM products to market quickly.

On-chip memory management and protection provides two key advantages for XENIX 3.0 over other microprocessor UNIX implementations. First, on-chip memory management and protection drastically reduces the overhead in accessing system memory as compared to the usual separate memory management unit. With this functionality right on the chip, the operating system works more smoothly and efficiently.

Second, on-chip memory management and protection circuitry ensures that each version of XENIX 3.0 will be very compatible with every other version. This heretofore impossible level of compatibility aids OEM, software developers, and end users due to the wider availability of compatible software.

Advanced microprocessor architecture provides pipeline processing, wherein a continual flow of instructions is kept in the CPU queue, results in throughput several times faster than the fastest competing microprocessor.



XENIX 3.0: Leading Edge UNIX Performance on a Micro

As the first UNIX operating system derivative optimized for the iAPX 286, XENIX 3.0 alone can take full advantage of the 80286's unique features:

Fast floating point processing is due to XENIX 3.0 support of the Intel iAPX 287 math coprocessor. Floating point processing delivers throughput that is an order of magnitude faster than non-floating point processing. Extra high processing speeds are needed in applications such as data base processing, commercial data reduction and graphics.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

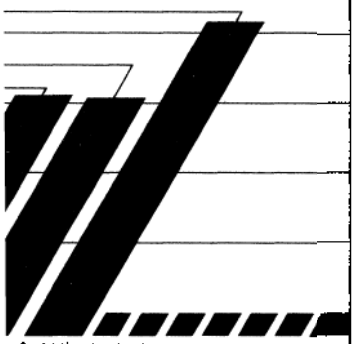
Faster, More Reliable Still When Teamed with Other Intel Systems Components

The throughput enhancements in the XENIX 3.0 software are pushed to even greater speeds by special hardware architecture in Intel's systems and board products.

MULTIBUS® System Architecture is the industry-standard system bus. It accommodates any of the special-purpose Intel iSBC® boards, as well as a multitude of third party Multibus boards and standard peripherals, for easy system expansion.

iLBX™ (Local Bus Exchange) is an Intel hardware innovation that increases the amount of local memory accessible by the operating system to significantly improve system throughput.

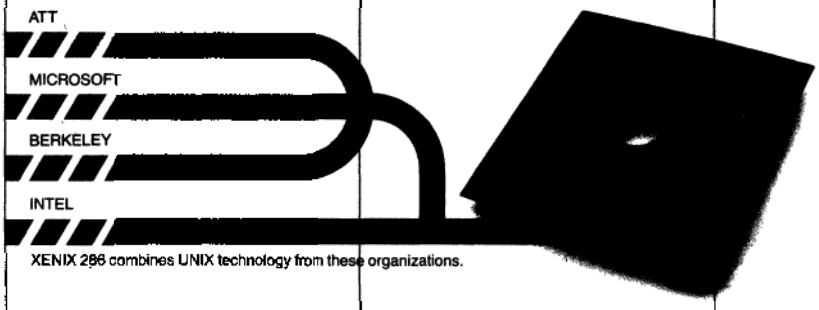
Error Correction Circuitry (ECC) automatically detects and corrects soft errors in RAM. This on-board, self-correction facility reduces errors and further underscores data integrity.



See Intel benchmark series

A Faster Operating System Means Market Leadership

The combination of the industry's most widely accepted operating system for multi-user, interactive applications with the industry's fastest and most advanced microprocessor gives the OEM a far superior price/performance ratio than is available through other options. The result for the OEM: market leadership due to the ability to more attractively price products based on superior performance.



XENIX 286 combines UNIX technology from these organizations.

XENIX 3.0: The Best of Everything

The XENIX 3.0 Operating System contains the best of many vendors' UNIX/XENIX development efforts during the last ten years (see Fig. above). We have taken the best features of many UNIX versions—ease of use, flexibility, performance, security, reliability—and added our own enhancements (not the least of which is compatibility with the iAPX 286) to make XENIX 3.0 the optimum software foundation for the commercial OEM.

Superior Data Reliability and Integrity

XENIX 3.0 contains enhancements to provide extremely high data reliability and integrity, particularly important to the OEM who is adding value to a system product. The following enhancements in XENIX 3.0 contribute to uniformly reliable data at all stages of application development.

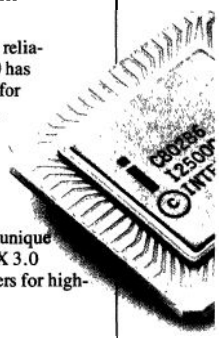
Automatic disk recovery is an improvement of the UNIX file system that allows automatic recovery of the file system in the event of unexpected system shutdown.

Record and file locks arbitrate multiple-access requests to the same record or file, allowing the programmer to extend locks to a single record, group of records or the entire file. This is important in multi-user applications to prevent two or more users accessing and updating the same information simultaneously.

XENIX System Analysis Test (XSAT) is a complete hardware-software diagnostic package included with all Intel integrated system products. XSAT provides a total analysis of a XENIX-based system, ensuring reliability even after the OEM configures new drivers into the system.

Tools for Easy System Configuration

In addition to increased data reliability measures, XENIX 3.0 has been functionally enhanced for easier system configuration. An interactive configuration utility allows the user to specify device drivers, disk buffers, memory size, etc., making it easy for the OEM to meet unique design requirements. XENIX 3.0 includes over 12 device drivers for high-speed controllers.



Friendlier Interface

The standard UNIX human interface has been enhanced in XENIX 3.0, with the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

addition of vi, a full-screen editor, for easier and faster application development.

The XENIX C shell augments the capabilities of the standard UNIX shell with the ability to maintain histories of invoked processes and provide the alias feature, saving re-keying of often-used commands. XENIX 3.0 also provides the visual shell, a menu driven command interpreter which makes full use of the screen to display status and environmental information to the user. It has a built-in HELP facility and allows users to add new applications to the menu.

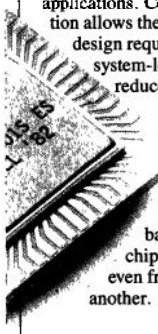
Intel's Open Systems Approach

Intel believes that system components—hardware or software—should be fully compatible with other family members at any level of integration and open to future VLSI advancements. XENIX 3.0 was designed to be part of the Open Systems concept.

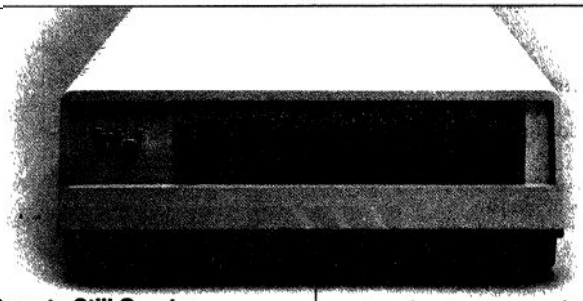
Portability from Chip to Board to System

Intel's XENIX 3.0 Operating System is available for and fully compatible across Intel component, board and system designs, something that no other XENIX version offers.

Such portability gives OEMs the flexibility to choose the most appropriate and profitable level of integration for their applications. Component-level integration allows the OEM to meet unique design requirements; board and system-level integration afford reduced time to market.



There is no loss in software development investment as your needs change, since you can port XENIX-based applications from the chip to the system level or even from one Intel processor to another.



Open to Still Greater Configurability through Third-Party Software and Hardware

XENIX 3.0 users can tap into an extensive base of existing third-party languages and application packages for almost endless versatility in system configurability. There are hundreds of such packages available today with many more on the way. To assure the availability and quality of these packages on our systems, we have the Independent Software Vendor Program. Through this activity, software vendors are given Intel systems as well as technical assistance to aid them in porting their packages. The resulting product is thoroughly evaluated by Intel prior to certification for operation on our system products.

Superior Documentation

In line with the OEM orientation of the Intel hardware and software combination, the documentation for Intel's XENIX 286 product provides excellent support for system builders. In addition to the mature UNIX documentation from AT&T and the value-added feature documentation by Microsoft, Intel adds a wealth of publications aimed at helping the OEM to successfully launch XENIX 3.0 based products.

Worldwide Support and Service

XENIX 3.0 customers can take advantage of Intel's worldwide staff of trained hardware and software engineers in contracting for application design assistance. A liberal warranty, including software updates and a technical newsletter, follows the sale. Once the warranty expires customers can choose from a variety of support contracts.

Intel offers complete training on the XENIX 3.0 Operating System as well as the iAPX 286 processor and associated hardware.

Intel, The Technological Leader...

Intel is committed to pushing the frontiers of VLSI design to their ultimate limits. In the process, we move our customers along the technology curve without interruptions in application development or expensive mid-stream architecture changes.

Intel started the micro revolution with the 4004 and has been the market leader with every generation of advanced processors since.

Systems and system software are a natural for us: who better knows the pieces and how to make them work together?

...In Total Solutions

The XENIX 3.0 Operating System fully exploits the iAPX 286, the fastest and most sophisticated microprocessor on the market. No other processor/operating system combination will give OEMs a faster and more economical path to getting systems and applications on the market.

Intel has always been first with the latest and most advanced VLSI and now with system software tailor-made for Intel VLSI. Because we're there first, our customers are first in their respective markets with state-of-the-art OEM and end-user products.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

XENIX 3.0 includes support for the following Intel Systems, single board computers and processors.

- System 286/310
- System 286/380

- iSBC® 286/10 Processor Board
  —16mb of addressing
  —On-chip memory protection
- CX Series RAM board
  —ECC (Error Correction Circuitry)
  —iLBX™ (Local Bus Extension)
- iSBC 215 Winchester Controller
- iSBX 218 Floppy Controller
- iSBC 534 Serial I/O Expansion Board
- iSBC 544 Intelligent Serial I/O Expansion Board
- iSBC 188/48 8-channel Serial I/O Expansion Board
- iSBC 552 Ethernet Controller Board
- iSBX 217 Tape Controller Board

- 80286 Central Processor
- 80287 Fast Floating Point Processor

Documentation

Documentation Includes:

- Overview of the XENIX 286 Operating System
- XENIX 286 Installation and Configuration Guide
- XENIX 286 System Administrator's Guide
- XENIX 286 Communications Guide
- XENIX 286 Visual Shell User's Guide
- XENIX 286 User's Guide
- XENIX 286 Reference Manual
- XENIX 286 C Library Guide
- XENIX 286 Programmer's Guide
- XENIX 286 Device Driver Guide
- XENIX 286 Text Formatting Guide

Text Books

The UNIX Book—Banahan & Rutter
The UNIX System—Bourne
The UNIX Operating System—Kaare
Understanding UNIX: A Conceptual Guide—Groff & Weinberg
The UNIX Programming Environment—Kernighan & Pike
Introducing the UNIX System—McGilton & Morgan
A Practical Guide to the UNIX System—Sobell
A User Guide to the UNIX System—Yates & Thomas
A Business Guide to the UNIX System—Yates and Emerson

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

Ordering Information

XNX 286 HRO	XENIX Object Software (8" double side, double density) plus license rights
XNX 286 KRO	XENIX Object Software (5¼" double-sided, double density) plus license rights
XNX 286 RF	Software Incorporation Fee
SYS 310-17X	System Kit including System 310-17 and XENIX Software
SYS 310-17MX	System Kit including System 310-17, XENIX Software, 6 user support
SYX 286 RO	License rights extension for system customers
SYX 286 RF	System incorporation fee

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Ordering Information

XNX 286 H	XENIX Object Software (8" double side, double density)
XNX 286K	XENIX Object Software (5¼" double-sided, double density)
XNX 286 RO	Software License Rights Extension
XNX 286 RF	Software Incorporation Fee
173258	XENIX Documentation Package
CTW 14PP	XENIX Customer Training
SPRTECHREP	XENIX Support Subscription Services
HOTLINE	XENIX Hotline Phone Service
SP86 330 XINSTALL	XENIX Software Installation
CONSULT-FIELD	XENIX Onsite Field Consulting
CONSULT-LT	XENIX Onsite Field Consulting for extended time periods.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

APPLICATION NOTE

AP-130

March 1982

Using Operating System Firmware Components to Simplify Hardware and Software Design

John Wharton
Microprocessor Applications

© INTEL CORPORATION, 1982

MARCH 1982
ORDER NUMBER: 210295-001

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INTRODUCTION

Intel recently introduced a new set of extensions to its microprocessor product line. The iAPX 86/30 and iAPX88/30 Operating System Processors (OSPs) augment the general-purpose instruction set of the well-known 8086/8088 architecture to include common, real-time, operating system capabilities. A single device, the 80130 Operating System Firmware component (OSF), now provides hardware support for functions previously relegated to software.

The 80130 introduces new concepts in the areas of both hardware and software. At first glance, traditional component-level hardware designers could feel somewhat intimidated by the esoteric concepts and unfamiliar buzzwords encountered in the software world. Even the experts in conventional operating system (OS) design may initially find it strange that what used to be "soft" software routines are now cast in silicon.

This application note is intended for readers at both levels. The first section reviews the development of processor extensions in general and operating system firmware in particular. Later sections should help you understand what a real-time operating system can do, how the 80130 provides these capabilities, and how to design system hardware and software to take advantage of such features.

The note also documents a complete (albeit simple) system, including schematics and listings. The reader may wish to reconstruct this system to get started with OSPs. Finally, a step-by-step description of the so-called "configuration" process shows how physical system parameters are incorporated into the software as the software is "installed" in memory. Throughout the note are a number of "exercises"—questions relating to concepts just presented. Please take a few moments to think about these questions before reading on.

The reader need not have worked with operating systems previously, though such background would be helpful. The reader should also know something about microprocessor hardware—at a minimum, how the 8086 or 8088 devices operate. For simplicity, most of the software examples are written in PL/M-86, so the reader should be familiar with PL/M-80 or some other block-structured language. Finally, be forewarned that the configuration steps make use of several ISIS utility programs, including EDIT, SUBMIT, ASM86, LINK86, and LOC86. Readers who wish to brush up on any of the above should consult the appropriate Intel reference manuals.

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

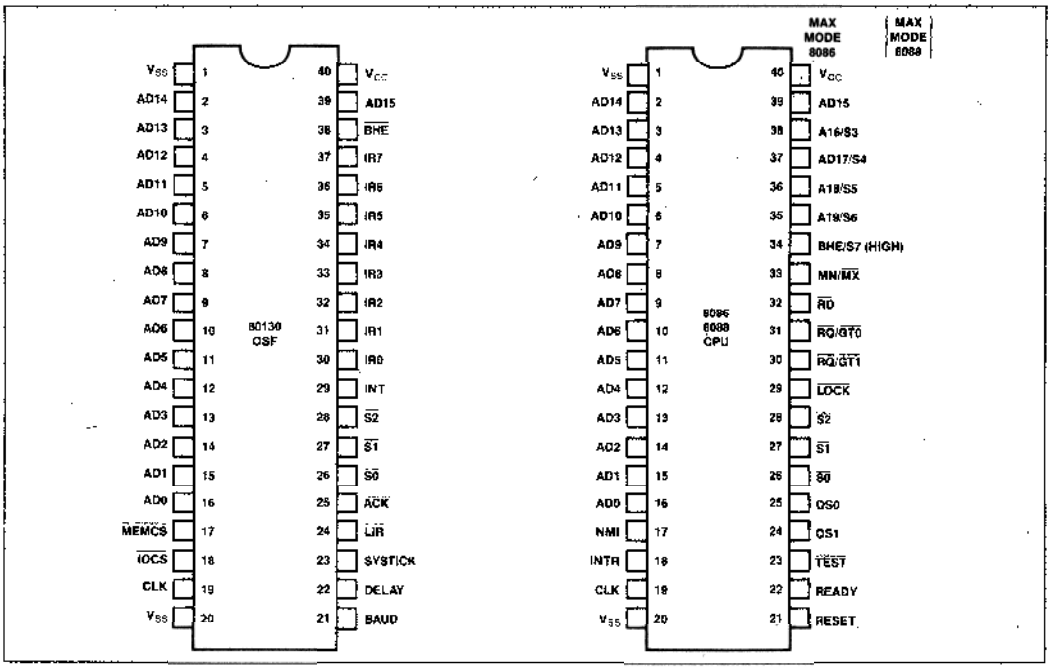


Figure 1. 8086 and 80130 Pinout Diagrams

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EVOLUTION OF PROCESSOR EXTENSIONS

In the early days of microcomputing (circa 1974), things were simple. The first microprocessors comprised just the central processing unit of a simple computer. Systems built up from these processors were generally small, dedicated-purpose device controllers—often replacing the random logic of an earlier design. The system designer had responsibility for the development of the hardware and all application software.

Semiconductor technology has progressed rapidly since then. Devices have become more sophisticated, as have the applications in which they are used. System functions today are more complex than they used to be, and are demanding more in the way of both system hardware and software.

To help designers cope with this complexity, semiconductor vendors are building increasingly more "functionality" into their standard product lines. Whereas the general arithmetic functions of the 8080 and 8085 were limited to addition and subtraction of eight-bit unsigned (ordinal) values, for example, the Intel® 8088 and 8086 now add, subtract, multiply, or divide eight- or 16-bit, signed or unsigned variables —an obvious improvement.

The evolution of floating-point arithmetic provides another example of technology growth. Initially, designers of numeric and process-control systems each developed the floating-point arithmetic routines they needed. Intel eased this task considerably in 1977 when it introduced a standard floating-point format and a floating-point arithmetic software library, FPAL-80. In 1978, the iSBC 310 High-Speed Mathematics Unit implemented these same functions with dedicated hardware and executed them an order-of-magnitude faster.

The 8231A Arithmetic Processor Unit (introduced in 1979) provided similar functionality in one chip at much lower cost. To accommodate the needs of today's world, the Intel RealMath™ software standard and the 8087 numeric coprocessor perform 80-bit floating-point arithmetic for high-performance 8088 and 8086 systems.

This evolution of floating-point hardware illustrates two recurring themes in the microcomputer industry. First, there is a natural trend toward componentization:

1. New applications reveal a need for new types of functionality (in this case, floating-point arithmetic).
2. As common requirements become evident, vendors develop software to serve these needs.
3. Specialized hardware is developed to support the established functions more simply and effectively than software alone.

In time, everything ends up in silicon.

The second theme is this: different functions should be implemented in different ways to fit the customer's needs. "Universal" requirements—like 16-bit multiplication—are best incorporated into the CPU. Functions needed only by certain applications—like high-speed, extended-precision square roots—should be provided as optional Processor Extensions so that their expense is incurred only by those who need them. In keeping with this philosophy, Intel currently offers several processor extension products (see "What's in a Name?").

What's in a Name?

The 80130 Operating System Firmware (OSF) device is only the latest member of an extremely flexible family of Intel microprocessors. Its siblings include the 8086 and 8088 Central Processing Units (CPUs), the 8089 I/O Processor (IOP), and a floating-point math coprocessor, the 8087 Numeric Processor Extension (NPX). These individual standard components may be mixed and matched in numerous ways to create combinations optimized for widely varying applications.

To make it easier to discuss the most common configurations, Intel has defined an "Advanced Processor Series" (iAPX) numbering scheme, something akin to those used in the minicomputer and mainframe worlds. The 8086 CPU by itself, for instance, is called the iAPX 86/10. The 8086/8087 combination is dubbed the iAPX 86/20. An 8086/80130 pair has the name iAPX 86/30. The 8086, 8087, and 80130 together would form an iAPX 86/40.

When each of these combinations uses an 8088 in lieu of the 8086, each of the numbers above substitutes "88" for the "86". An 8088 teamed with an 80130 is therefore called the iAPX 88/30. Finally, adding an 8089 to any system changes the final zero to a one. So, an iAPX 88/41 system would be one using the 8088/8087/8089/80130 chip set.

Real-Time Operating Systems

Let's turn our attention now to the subject of microcomputer operating system software—an area steadily growing in importance. The trends toward standardized functions with specialized implementations will become evident.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

But first, what is an operating system? The phrase means different things to different people. In 20 words or less: An OS is a tool, a set of programs or routines which reduce and simplify the problem of managing system resources. (Well, 21, actually . . .)

Most microcomputer programmers have encountered single-user diskette operating systems, Intel's ISIS-II®, and CP/M® and CP/M-86® from Digital Research Incorporated among them. In essence, an OS of this sort is a collection of run-time subroutines which perform device I/O operations and give application programs access to a disk-based file system. Along with these are routines to supervise the loading and execution of application programs. Historically, this type of OS is oriented toward user-interactive applications: software development, business computing, and the like.

In the mainframe world, the goal of an operating system is to use expensive equipment as efficiently as possible. Batch processing systems ensure that programs waste as little CPU time as possible, though each monopolizes the CPU until it has completed. A time-sharing OS allots short periodic "slices" of time to each of several independent users, during which each has access to the CPU, memory, and other system resources.

A step above the traditional time-sliced OS are "real-time, multitasking operating systems." But what is a "real-time" application? ("Don't all programs execute in real time?")

A real-time system is one in which the CPU must do many different things (tasks), all more-or-less simultaneously. Unlike the sequential time-sharing of mainframe OSs, though, the tasks are prioritized. Low-priority tasks are preempted if any of higher priority have work to do. The higher-priority task then runs until it must wait for some external event to occur or no longer needs the CPU for some other reason. Thus, the CPU services tasks in their order of importance.

A computer controlling factory machinery, for instance, might perform five separate tasks:

1. Monitor input switches to detect emergency conditions, determine intended operating mode, or update indicator lights showing machine status;
2. Drive a stepper motor to position a tool;
3. Keep track of the time of day;
4. Send output to the console (e.g., CRT), either in response to explicit commands or as part of some other task;
5. Read and process characters entered from a console keyboard.

These tasks seem largely unrelated, though the first few may be more important to system operation than the others. Let's consider some alternate ways to accomplish these functions with today's microcomputers.

Conceptually, the most straightforward approach might be to dedicate a separate computer to each. The program for each would then be quite simple: an initialization phase followed by an endless loop performing the dedicated function. Algorithms for the first four tasks are flowcharted in Figure 2.

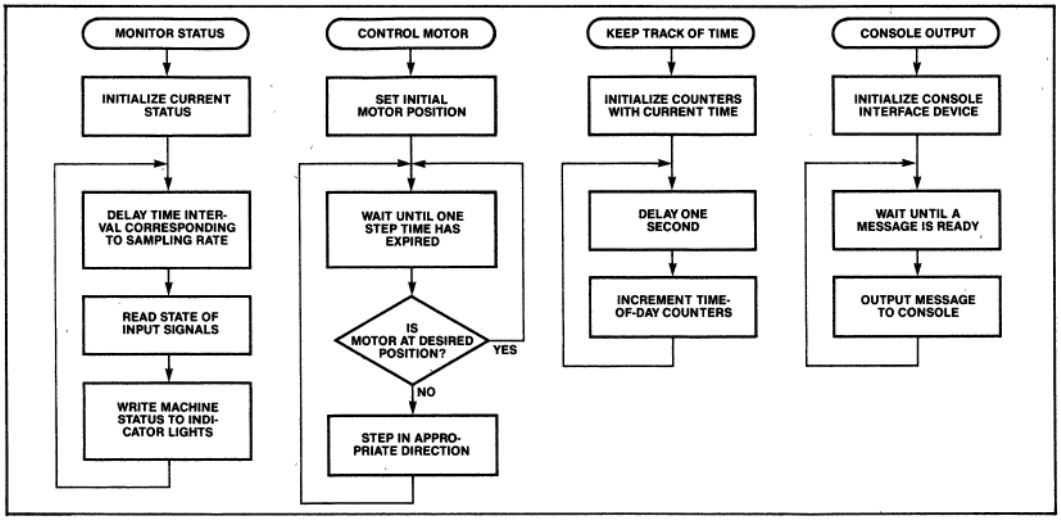


Figure 2. Flowcharts for Concurrent Machine-Tool Tasks

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

What's wrong with this approach? Ignoring cost, the need for multiple CPUs becomes physically unrealistic for more than a few tasks—60, say, or 600. And tasks are rarely fully independent; note that the switches monitored by task 1 could affect task 2, and that tasks 4 and 5 interact with the rest of the system in as yet undefined ways. So, some sort of communications would have to be set up between the micros.

Exercise 1. Suppose five tasks are all interrelated. How many communications channels would have to be set up between different processors? If each channel requires two dedicated communication chips, how would the number of peripheral devices compare with the number of CPUs?

In each task, the CPU spends most of its time waiting for time to pass or for something to happen. One CPU would be able to implement all five tasks if its time were properly divided among them. An alternate approach, then, might be for a single processor to attend to each task in turn, performing the actions called for by each. Figure 3 shows a flowchart for this scheme. Only one CPU is required and the tasks can communicate between themselves and share physical resources like the console.

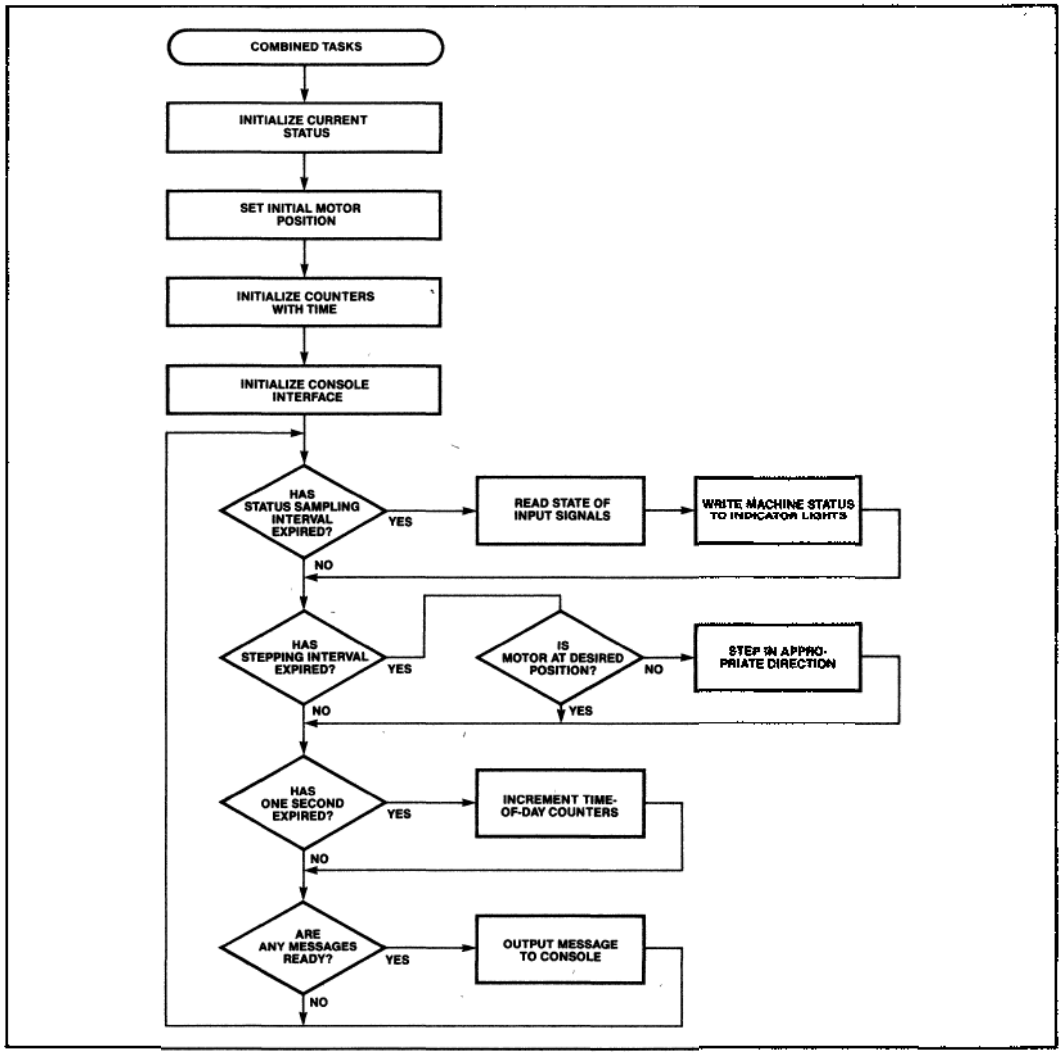


Figure 3. Machine-Tool Tasks Implemented Via Polling Scheme

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The problem here is the heavy interaction between tasks. Before it can be serviced, an important task may have to wait for many other less critical tasks to complete. This imposes a constraint that each task release the CPU as quickly as possible. Also, lumping tasks together obscures the boundaries between them. Initialization sequences must be grouped with each other, rather than with the sections of code affected. Adding to or deleting any task may affect the others. It's not clear how to structure the program such that programmers could cooperate on such a program.

Moreover, the various tasks can interfere with each other. Suppose on a given pass through the processor loop, three tasks each send one new character of a message to the console display screen. The resulting output would be most interesting.

The third, and optimal approach, would be one which combined the advantages of the first two approaches, while avoiding the pitfalls. Each function of the overall system could be designed, written, and tested separately, as in the first approach, yet all the software would run on a single computer system as in the second. Tasks could therefore communicate with each other easily, and share peripherals such as CRTs. This multitask control and communication function could be performed largely through software.

The key is finding a way to properly budget CPU time between the various tasks. Early pioneers of complex, real-time, control system design found that they needed special routines, apart from the application tasks themselves, to supervise the execution of application tasks. It was (at best) an inconvenience for so many engineers to independently define, design, document, test and debug software with the same general purpose. At worst, schedules slipped or projects were cancelled for the lack of reliable executive software.

To help avoid these hazards and free up the designers to concentrate on more immediate goals, Intel developed iRMX 80, the first real-time, multitasking, executive operating system for microprocessors. iRMX 86 was introduced to the 16-bit world two years later in 1980.

Because of the critical real-time nature of such operating systems, they require certain hardware capabilities in the host system, such as special timer logic clocked at certain frequencies to measure the passing of time, and interrupt controllers to monitor assorted asynchronous events. Combine all this with a handful of memory chips to house just the OS software, and the address decode and control logic needed by all of the above, and you'll find you need the equivalent of a single-board computer system just to support a multitasking environment.

Until now, that is. The current trend is to integrate OS software and hardware functions into silicon. Intel's iAPX 432 32-bit MicroMainframe™ system does this within the CPU. For the 16-bit world, however, Intel provides a separate chip, the 80130, which contains operating system firmware as well as timer and interrupt control functions.

What is the 80130 OSF? It is an extremely sophisticated integrated circuit, fabricated using Intel's high-performance HMOS technology, which contains over 160,000 devices. In one 40-pin package (Figure 4), the 80130 combines several timers, multiple-mode interrupt control logic, and a large control store memory —plus buffers, decoders and the like—to form the integrated heart of a multitasking operating system. Compared with the iRMX 86 Nucleus, for example, the 80130 replaces an 8259A PIC, an 8253 PIT, a special oscillator, 16K bytes' worth of memory, and associated control logic.

The 80130 operates in conjunction with the 8086 CPU. Together, the two chips are called the iAPX 86/30 OSP. The same device may be paired just as easily with an 8088 forming the iAPX 88/30. From here on, though, references to the 8086 or "host processor" apply to both CPUs. Due to the high speed of HMOS, the 80130 currently runs at system clock rates up to 8 MHz without inserting any wait states. Firmware in the 80130 supports the 35 primitive functions listed in Table 1. Many of these are discussed in Chapter IV.

SYSTEM HARDWARE DESIGN

The 80130 supports a wide range of system architectures, from compact to quite complex. Most, however, have in common the functional blocks represented in Figure 5. After a brief review of iAPX 86/30 systems in general, we'll examine 80130 requirements in greater detail.

Basic Functional Blocks

In addition to the 80130, the central processing "core" of a typical OSP system would include an 8088 or 8086 operating in maximum mode, an 82843A clock generator, and an 8288 system controller, all connected according to the standard rules. More on the 80130-specific interconnects later.

Address latches (e.g., 8282s or 8283s) are generally needed to demultiplex the processor address bus for standard memory devices and for memory and I/O device-select logic. The number (from zero to three octal latches) depends on the host processor, memories, and the addressing scheme employed. Data

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 1. Operating System Primitives Supported by 80130

Task Management Suspend Task Resume Task Sleep Create Task Delete Task Set Priority Get Task Tokens	**Interrupt Management** Set Interrupt Signal Interrupt Reset Interrupt Enter Interrupt Wait Interrupt Exit Interrupt Enable Disable Get Level
Intertask Communications and Synchronization Send Message Receive Message Create Mailbox Delete Mailbox	**Free Memory Management/System Partitioning** Create Segment Delete Segment Create Job
Mutual Exclusion Control Receive Control Accept Control Send Control Create Region Delete Region	**Misc. Support** Signal Exception Get Type Disable Deletion Enable Deletion Set O.S. Extension Get Exception Handler Set Exception Handler

transceivers (8286s or 8287s) may also be needed for increased bus buffering.

Any complete microprocessor system must also have some combination of I/O peripherals and memory, collectively indicated by the box labeled "Local Resources." As we shall see, some of the system RAM and ROM (or EPROM) must be reserved for OSP itself. Additional logic decodes the latched address lines to generate chip-select signals for the memory and I/O devices.

This note only discusses simple, single-processor systems. More sophisticated architectures may incorporate a multimaster system bus, in addition to a local processor bus. This would require additional system controllers, address latches, and bus transceivers for bus isolation, and address mapping logic (not shown) to select between the various busses, enable the respective transceivers, generate a System Ready signal, and so forth. For design information on such techniques, refer to application note AP-67 in the *iAPX 86,88 User's Manual*.

80130 Pin Functions

Back to the 80130. Certain pins on the 80130 (in particular, AD15–AD0) attach directly to the CPU. The AD pins are bidirectional, accepting addresses from the host and returning instructions or data. By monitoring the system clock and status signals, $\overline{S2}$–$\overline{S0}$, the 80130 can decode the processor status internally and respond automatically to the appropriate bus cycles. The $\overline{BHE}$ input lets the 80130 determine the width of data transfers and distinguishes an 8088 host from an 8086. If you refer back to Figure 1, you'll notice that these 80130 pin assignments were selected to simplify P.C. board layout.

Because of the 80130's location on the CPU side of any latches or data transceivers (on what is sometimes called the "pin bus"), the transceivers (if used) must be disabled when the 80130 is driving the processor bus. Whenever the 80130 is responding to any type of bus cycle, it generates an $\overline{ACK}$ signal. As Figure 4 suggests, one way to avoid contention is to simply disable the transceivers when $\overline{ACK}$ is active. $\overline{ACK}$ can also be used to prevent the insertion of wait states.

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

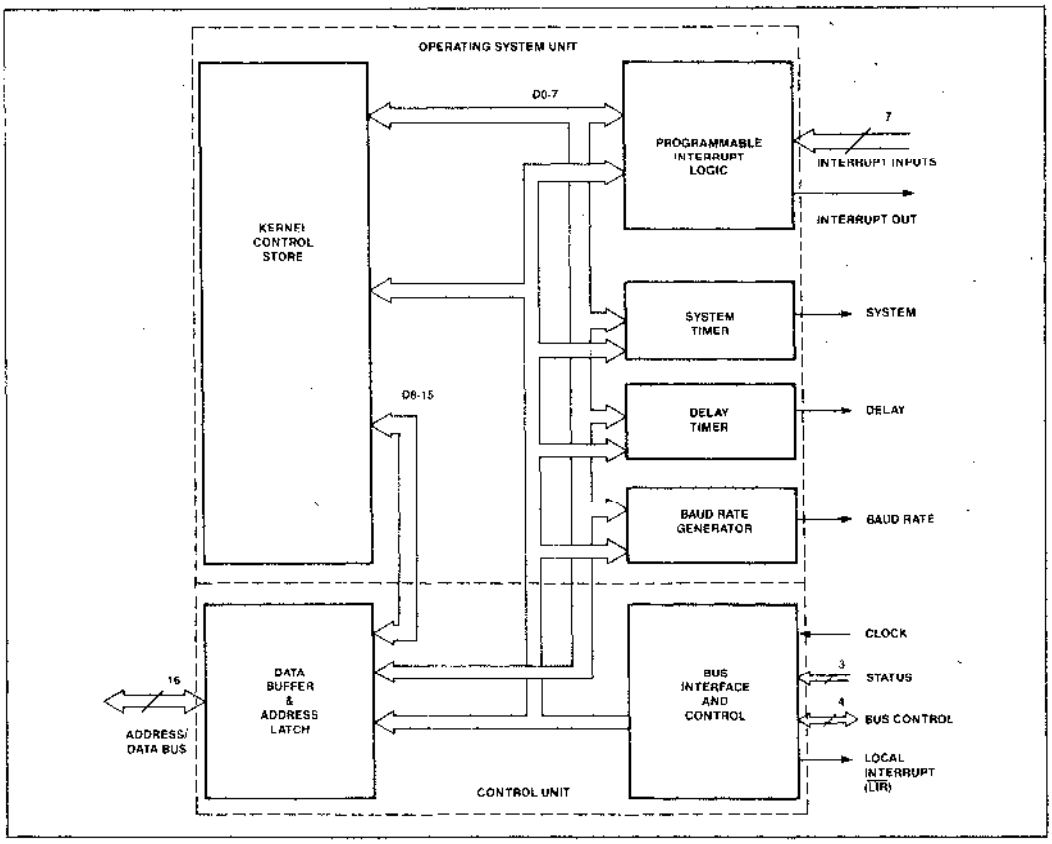


Figure 4. 80130 Internal Block Diagram

Additional pins on the 80130 include eight interrupt-request inputs. Internal interrupt control logic provides many of the functions of the 8259A. During system configuration (Chapter V), each of the eight may be individually defined as a direct level-sensitive or edge-triggered interrupt request, or each may be cascaded with a standard 8259A in slave mode.

The INT output must be connected to the host CPU to inform it of an enabled interrupt request. In very large systems with multiple, cascaded interrupt controllers, Local Interrupt Request ($\overline{LIR}$) indicates to the bus contention logic whether a requesting slave is local, or must be accessed via a multimaster bus.

The 80130 also contains dedicated timer logic to provide the OS time base, which is output on SYSTICK. Software operating in conjunction with the 81030 assumes one of the interrupt inputs (INT2 in this case) is driven by SYSTICK, so this connection must be made externally. Routines within the 80130 initialize and perform all bit-level control of the interrupt and timer logic, according to options and parameters specified during the configuration process. Freeing the programmers from this tedium allows them to devote more thought to solving their own unique problems.

An additional, independent timer generates a user-programmable, square-wave output signal called BAUD to clock an off-chip USART.

Since the 80130 displays some of the characteristics of both memory and I/O, it requires chip-select signals for both the memory ($\overline{MEMCS}$) and I/O ($\overline{IOCS}$) address spaces. These are discussed at length below. Finally, Intel has reserved one output pin (called "DELAY") for use in future designs. Leave it unconnected in iAPX 86/30 systems.

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

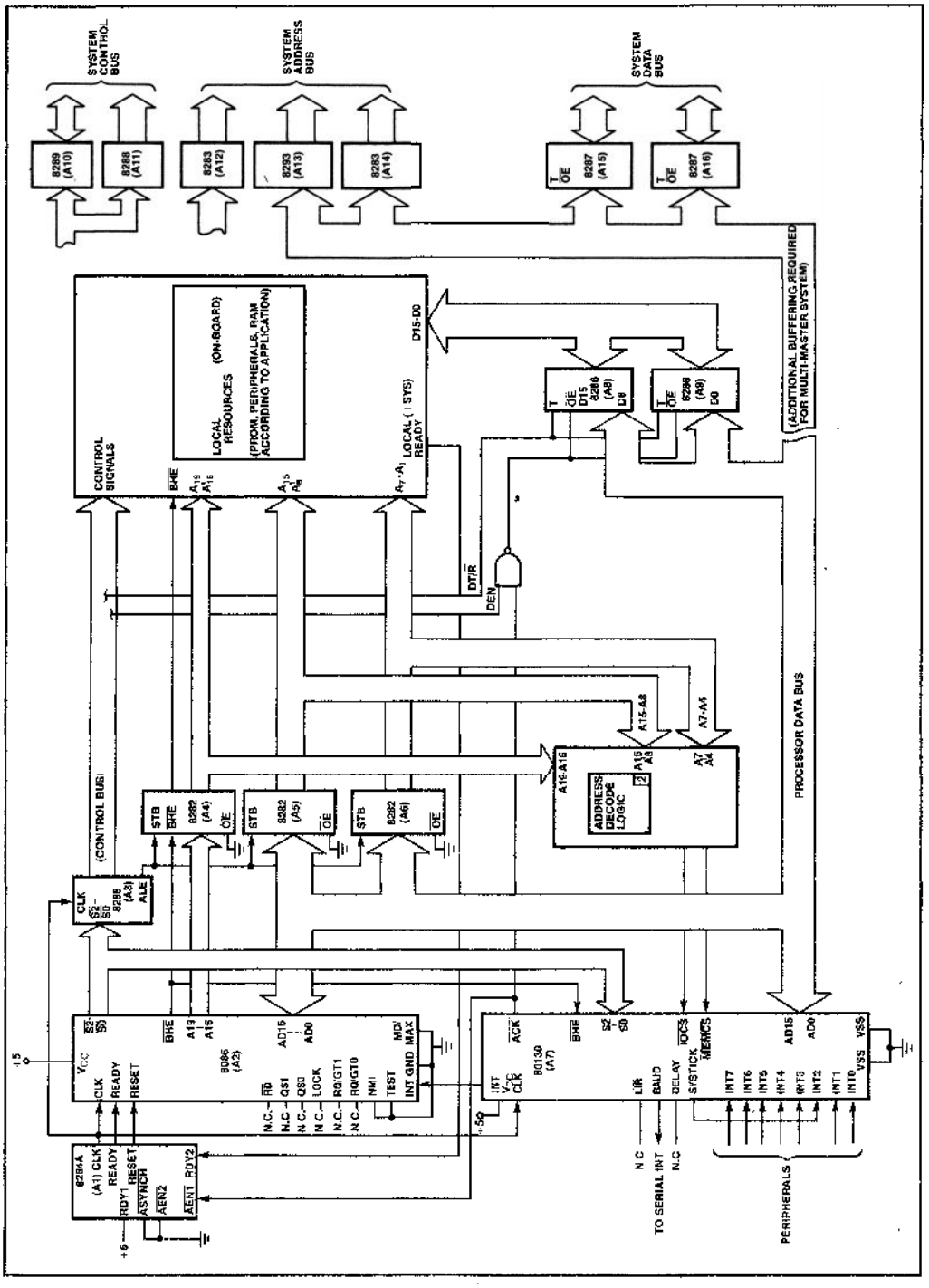


Figure 5. Basic iAPX 86/30 Microcomputer System Block Diagram

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Additional System Requirements

The OSP requires a certain amount of off-chip memory for its own operation. The system must provide at least 1K bytes of RAM at address 00000H for the CPU interrupt vectors, plus another 1500_{10} bytes for OSP system variables, data structures, stacks, and the like. This RAM may reside anywhere in the 8086 megabyte address space, although it is often contiguous with the interrupt vector up front. Application tasks must each have their own stack, so allow at least an additional 300 bytes of RAM for each.

Any iAPX 86 system must have ROM or EPROM at the upper end of memory to hold the CPU restart vector. About 3400 more bytes are consumed by code to initialize and access the OSP. This code is generated automatically from libraries on a diskette provided with a product called the iAPX 86/30 and iAPX 88/30 Operating System Processor Support Package (iOSP 86). Space left in the initialization EPROMs is available for application tasks.

As code is being written, the system designer should count on another 1500 bytes of code from the support libraries being added to his application during the linking and system configuration steps. These memory requirements are shown in Figure 6. In practice, the separate blocks in this figure would be grouped together for more efficient use of RAM and EPROM chips.

The 80130 occupies a 16K-byte block of addresses in the host-processor memory space, so external logic should decode address bits A_{19}–A_{14} to generate $\overline{\text{MEMCS}}$. Similiarly, the timer and interrupt control logic occupy a 16-byte block of addresses in the I/O space; at least some of the bits A_{15}–A_4 must be decoded to generate $\overline{\text{IOCS}}$. The 80130 decodes all the lower-order address bits (14 for memory, four for I/O internally).

Firmware in the 80130 leaves a great deal of flexibility in decoding the chip-select signals, to be compatible with whatever decode logic is already present in the system. The I/O starting address may be on any 16-byte boundary in the full CPU I/O space. The memory block has only two restrictions: the off-chip initialization and interface code memory must be placed immediately above the MEMCS block, so the 80130 may not occupy the extreme top of memory, nor may the 80130 reside at address 00000H since this area is reserved for interrupt vectors.

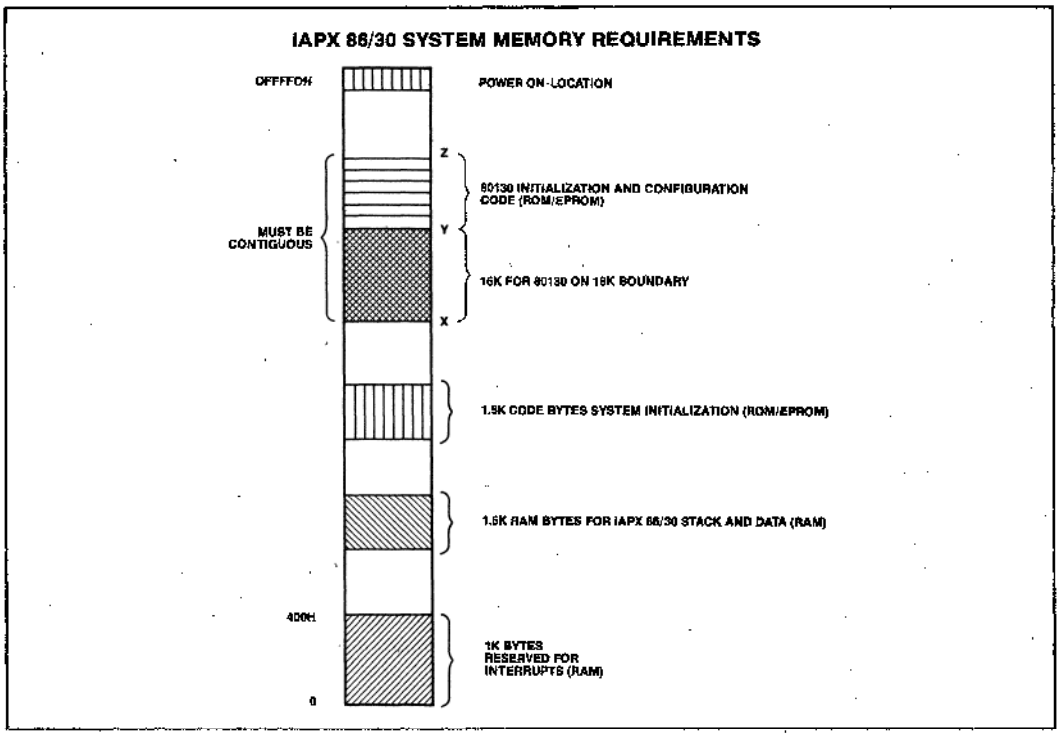


Figure 6. Operating System Processor System Memory Requirements

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Timing Requirements

System timing analysis is often the most tedious part of digital hardware design. This discussion can be relatively short, though, because the 80130 timing is quite simple: by design, the part is compatible with the timing of the host processor. Since it interfaces directly with the CPU pins, traditional set-up, hold, and access times no longer matter.

There are really only two areas of concern in analyzing the timing of most OSP systems, both of which relate to the user-generated chip-select signals. Figure 7 illustrates the relevant timing signals of a standard 8086 four-state Read cycle (memory or I/O), along with the timing responses of the 80130. I/O Write cycle timing is the same. (Full timing diagrams are part of the respective data sheets.)

The first concern is that $\overline{\text{MEMCS}}$ and $\overline{\text{IOCS}}$ must be active early in a memory or I/O cycle if the 80130 is to respond during T_3. In each case, the chip-select signals must be active T_{CSCL} before the end of state T_2. Assuming wait states aren't desired, addresses generated by the CPU must propagate through the address latches and be decoded during T_1 or T_2.

How much time does this leave the decode logic? As we'll see, ample.

By convention, T_{CLAV} is the delay from the start of T_1 until address information is valid on the CPU pins; T_{IVOV} is the propagation delay through an 8282 latch; and T_{CSCL} is the 80130 chip-select set-up time. The mnemonic T_{OVCS} represents the chip-select logic propagation delay, after the latch outputs are stable. The sum of these four delays must be less than two system clock cycles, reduced by the clock transition time.

$$T_{CLAV} + T_{IVOV} + T_{OVCS} + T_{CSCL} \leq T_{CLCL} + T_{CLCL}$$

$$T_{OVCS} \leq T_{CLCL} + T_{CLCL} - T_{CLAV} - T_{IVOV} - T_{CSCL}$$

$$\leq 125 + 125 - 60 - 30 - 20 \text{ (nsec.)}$$

$$\leq 140 \text{ nsec.}$$

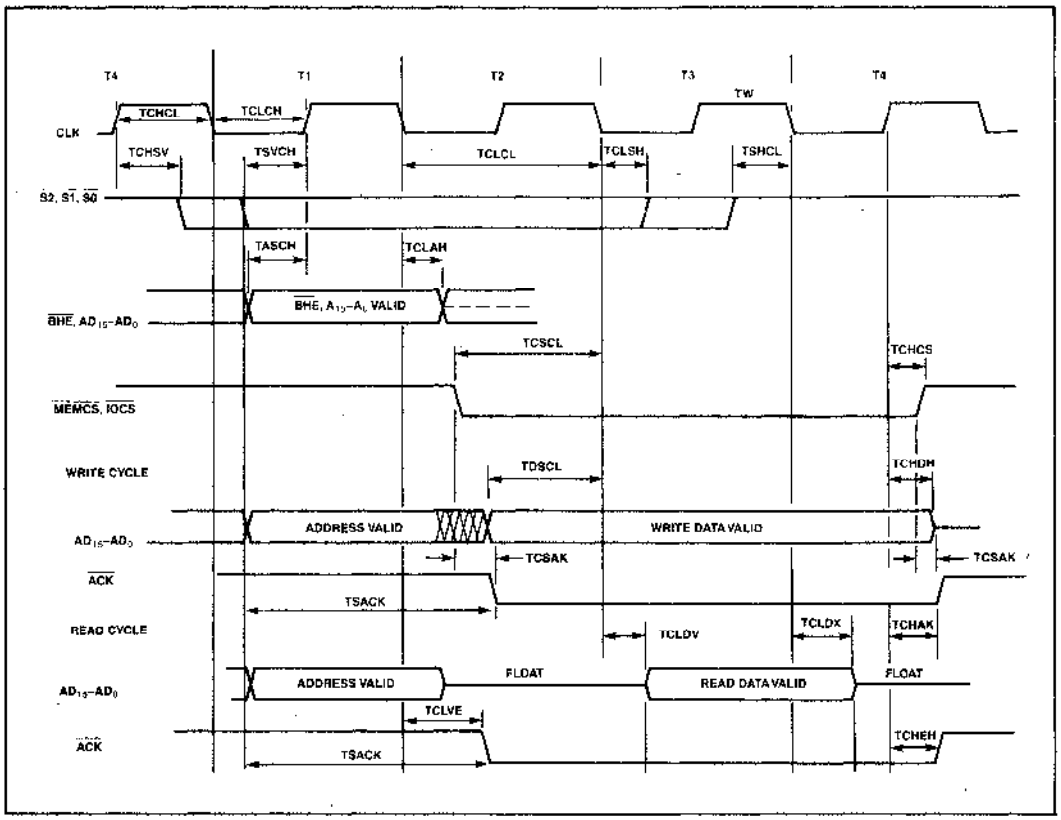


Figure 7. Operating System Processor Timing Diagrams

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The propagation delay numbers plugged into the equation are worst-case values from the appropriate Intel data sheets. The CPU is an 8086-2 operating at 8 MHz. This means the address decode logic must produce stable CS outputs within 140 nanoseconds.

Exercise 2. Using standard, low-power Schottky TTL, does it make sense for a circuit to take longer than 140 nsec. to decode 6 program or 12 I/O address bits? Even if the rather liberal setup specs are not met, the 80130 would still work fine. Wait states would be needed until the chip-select signal was active, however, so performance would degrade some.

The second point of concern relates to ready signal timing. The 80130's acknowledge output signal, $\overline{ACK}$, can be used to control the CPU's ready signal. For this case, the chip-select signal must be active early in a memory or I/O cycle to allow activation of $\overline{ACK}$ early enough to prevent wait states. There are two schemes for implementing ready signals; "normally ready" and "normally not ready." (For more details, refer to AP-67, "8086 System Design.") Chip-select timing is more critical in some "normally not ready" systems.

In a "normally not ready" design, acknowledge signals are generated when each resource is accessed. The individual acknowledgements are combined to form a system-wide ready signal which is synchronized by the 8284A clock generator via the RDY and AEN inputs. The 8284A can be strapped to accept asynchronous ready signals (asynchronous operation) or to accept synchronous ready signals (synchronous operation). Synchronous 8284A operation provides more time for address latch propagation and chip-select decoding. In addition, inverting ACK off chip produces an active-high ready signal compatible with the 8284A RDY inputs, which have shorter set-up requirements than AEN inputs. (As a side benefit, a NAND gate used like this can combine ACK with the active-low acknowledge signals from other parts of the system.) Based on these assumptions, the time available for address latch propagation and chip-select decoding at 8 MHz is:

$$T_{CLAV} + T_{OVCS} + T_{CSAK} + R_{HVCL} \leq T_{CLCL} + T_{CLCL}$$
$$T_{OVCS} \leq 2\,T_{CLCL} - T_{CLAV} - T_{CSAK} - T_{HVCL}$$
$$\leq 250 - 60 - 110 - 35$$
$$\leq 45 \text{ nsec.}$$

The circuit in Figure 8 which uses Schottky TTL components leaves about 15 nsec. to produce $\overline{MEMCS}$ from

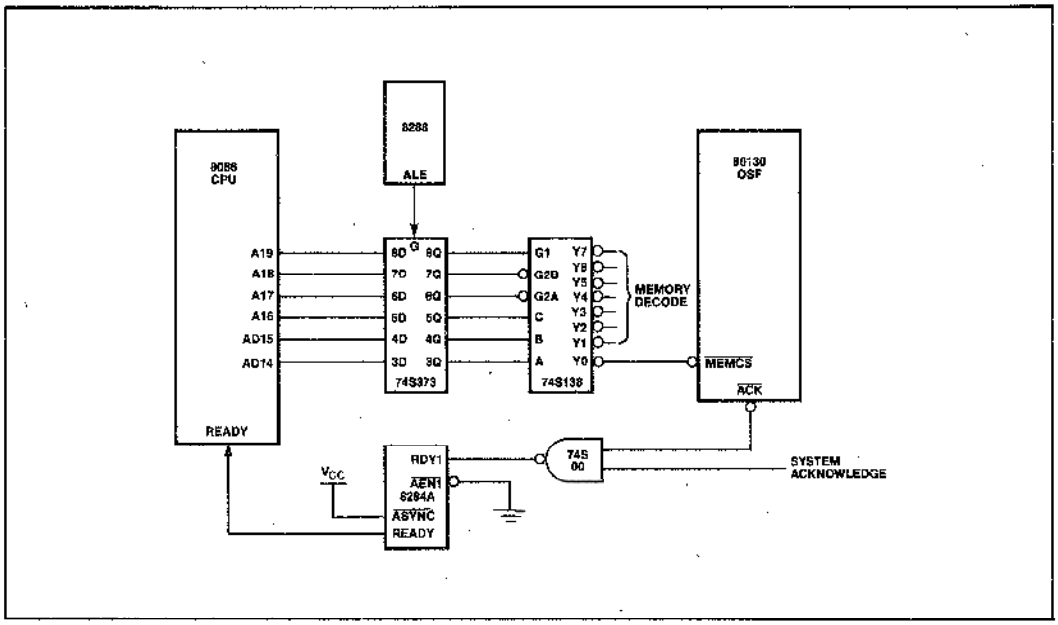


Figure 8. High-Speed Address Decoding Circuit

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the high-order address bits—more than enough for the 74S138 one-of-eight decoder shown.

Granted, this does not leave much leeway to fully decode the I/O address bits. A 12-input NAND gate on AD15–AD4 could be used, introducing only a single propagation delay but forcing the I/O register block to start at 0FFF0H. Incomplete decoding is also legal: it is safe to drive $\overline{IOCS}$ with the (latched) AD15 signal directly, provided all other ports in the system are disabled when this bit is low. In this case, the effective address of the I/O block (which must be specified during the system configuration step) could be 0000H, or any other multiple of 16 between 0000H and 7FF0H.

Again, the OSP system will still operate even if the memory or I/O decoding is slow. The acknowledge signal returned to the host CPU would just be delayed accordingly, so unnecessary wait states would be inserted in access cycles, but the 80130 would *not* malfunction. Only rarely does the OSP access resources in its I/O space. Even if slow decode logic were to insert several wait states into every I/O cycle, the overall effect on system performance would be insignificant.

A few words of caution, though. If the 8284A is strapped for synchronous operation, external circuitry must guarantee that ready-input transitions don't violate the latch set-up requirements. Also, the chip-select signal must *not* remain low so long after the address changes that the 80130 could respond to a non-80130 access cycle.

Exercise 3. Suppose the typical timing values for a particular decoder would easily meet the ready-input set-up requirements presented above for asynchronous 8284A operation, but pathological worst-case figures were just a little slow. Could that circuit still be used safely in most applications? What would happen if the worst-case combination of worst-case conditions ever actually did occur? These occasional extra wait states would probably not cause a hard system failure.

Exercise 4. Earlier it was mentioned that the acknowledge signal could also be used to avoid bus contention. Prove that with any decode logic which meets the above requirements, $\overline{ACK}$ would disable the bus transceivers before the host CPU samples the bus.

Example System Design

Appendix A includes full schematics for a complete iAPX 86/30 system providing considerable functionality with only 27 chips. In addition to the OSP, the system has 4K bytes of 2114 RAM (with sockets for another 4K), from 8K to 32K bytes of 2732A or 2764 EPROM, an 8251A USART operating at 9600 baud, and an 8255A Programmable Peripheral Interface with 24 parallel I/O lines. Eight of the inputs read logic values off DIP switches; eight outputs drive small LEDs. Four more outputs connect to the coil drivers of a four-phase stepper motor. A layout diagram of the prototype appears in Figure 9.

The system is even simpler than the discussion of "typical" requirements implied. The 8086 direct-bus drive capability is adequate to make the data transceivers unnecessary. (To equalize the bus loading, the 8255A is connected to the upper half of the bus.) Address decoding logic was minimized by making the high-order address bits "don't-cares." Moreover, the part count could have been reduced to 16 using an 8088 and multiplexed-bus 8185 RAMs and 8755A EPROMs. (The reader may be surprised to learn that, except for wire-wrapping mistakes, the prototype system hardware worked when it was first powered up. The author certainly was!)

APPLICATION SOFTWARE DEVELOPMENT

Like other well-structured programs, application software to run on the iAPX 86/30 is written as a number of separate procedures or subroutines. In conventional programs, though, execution begins with a section of code (the *program body*) at the outermost level. The program calls application procedures, which may call other procedures, but which eventually run to completion and return to the program body.

In an OSP application, though, there is no "outermost level" in the traditional sense; rather, the procedures are started, suspended, and resumed as situations warrant under the control of the OSP. The term "task" refers to the execution of such a procedure in this way. While an instruction stream is suspended, the OSP keeps track of the task state (instruction counter, CPU register contents, etc.) so that it may be resumed later.

Each task is assigned a relative priority by the programmer, on a scale of 0 (high priority) to 255 (low). Tasks with higher (numerically lower) priority are given preferential treatment by the OSP; the task actually controlling the CPU at any given instant will be the one with the highest priority which is not waiting for some event to occur. (If all this sounds confusing, examples coming later may help.)

A task which operates independent of other tasks can be written without knowing anything about the others.

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

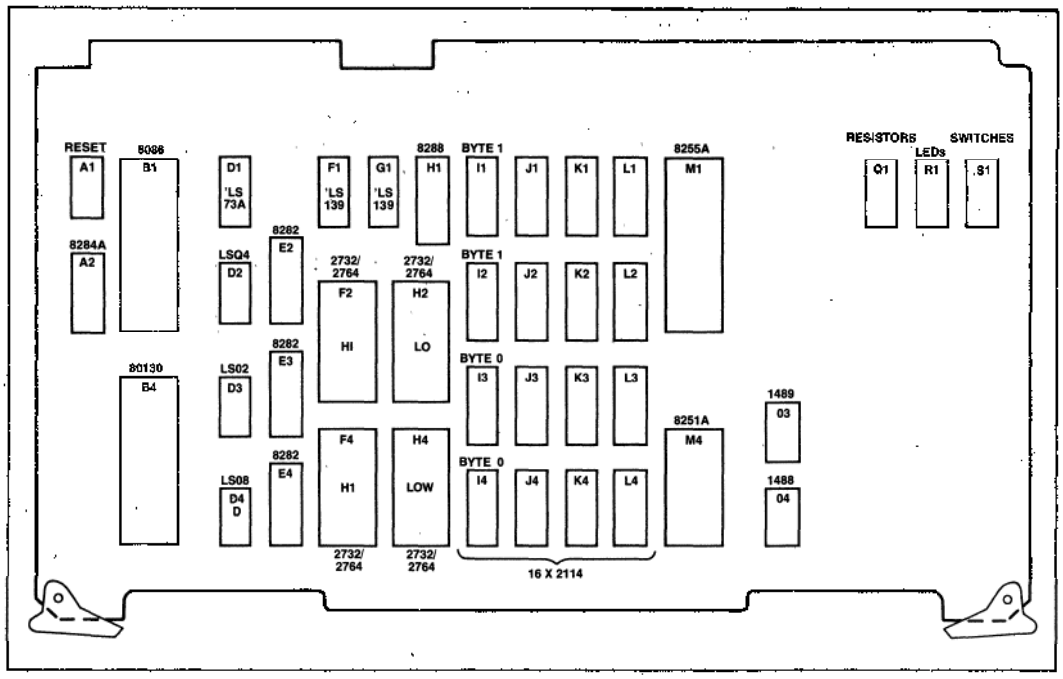


Figure 9. Example System Prototype Layout

This makes it easy to divide a very large programming job among a team of programmers, each writing the code for some of the tasks. Moreover, a task need not even know if other tasks exist. They may be tested and debugged before others have even been written. As an application evolves, new tasks may be added or unnecessary ones removed without affecting the rest.

The number of tasks in an application may need to be quite large. The number of tasks allowed in one application is essentially unlimited, as is the number of other objects—regions, mailboxes, segments, and the like. (The term "object" relates to different types of data structures maintained internally by the OSP.) Each object is internally identified by a unique 16-bit "token," which means the theoretical maximum total is over 65,000. The more pragmatic issue of physical memory consumption limits the number of simultaneous concurrent tasks to "only" several thousand.

(When a number of tasks cooperate to accomplish some common goal, the collection of tasks is referred to as an application "job." The OSP also allows for an unlimited number of application jobs, though only one is illustrated in the example discussed here. A second similar machine, with different status switches, a different motor, and a different console might make up a second job.)

All OSP application jobs must have one special initialization task (often called INIT$TASK) just to get started; this one may, in turn, create other tasks as it executes. The initialization task for this example is discussed at the end of this chapter.

Hardware Initialization

The life of any task can be broken into three phases: start-up, execution, and termination. The start-up phase initializes variables, data structures, and other objects needed by the task. During the execution phase the task performs its useful work. Depending on the application, this may be a single sequence of actions, or a loop executed repeatedly. When the task completes, it must terminate itself so as not to use any more CPU time. One or more phases may be omitted. For example, some tasks are intended to execute "forever," in which case the termination phase is not required.

This life cycle is suggested by Example 1, a segment of code called HARDWARE$INIT$TASK. This task first

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

programs the 80130 internal timer logic to generate a square-wave cycle on the BAUD pin every 52 system clock cycles, which corresponds to a system console data rate of 9600 baud. The task then sets the system's 8255A PPI and 8251A USART devices to operate in the desired modes, and outputs a short sign-on message to the CRT. For the sake of reader's unfamiliar with the protocol for interfacing with the 8251A, simple input and output routines (C$IN and C$OUT) are reproduced in Example 2.

```
HARDWARE$INIT$TASK:  PROCEDURE,
    DECLARE HARD$INIT$EXCEPT$CODE WORD,
    DECLARE PARAM$51 (*) BYTE DATA (40H, 6DH, 00H, 40H, 4EH, 27H),
    DECLARE PARAM$51$INDEX BYTE,
    DECLARE SIGN$ON$MESSAGE (*) BYTE DATA
        (CR, LF, 'iAPX 86/30 HARDWARE INITIALIZED', CR, LF),
    DECLARE SIGN$ON$INDEX BYTE,

    OUTPUT(PPI$CMD)=90H,
    OUTPUT(TIMER$CMD)=0B6H,
    OUTPUT(BAUD$TIMER)=33.   /*GENERATES 9600 BAUD FROM 5 MHZ*/
    OUTPUT(BAUD$TIMER)=0,
    DO PARAM$51$INDEX=0 TO (SIZE(PARAM$51)-1),
        OUTPUT(CMD$51)=PARAM$51(PARAM$51$INDEX),
        END.    /*OF USART INITIALIZATION DO-LOOP*/
    DO SIGN$ON$INDEX=0 TO (SIZE(SIGN$ON$MESSAGE)-1),
        CALL C$OUT(SIGN$ON$MESSAGE(SIGN$ON$INDEX)),
        END.    /*OF SIGN-ON DO-LOOP*/
    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @HARD$INIT$EXCEPT$CODE),
    CALL RQ$DELETE$TASK(0, @HARD$INIT$EXCEPT$CODE),
    END  HARDWARE$INIT$TASK,
```

Example 1. System Hardware Initialization Task

```
C$OUT:  PROCEDURE (CHAR),
    DECLARE CHAR BYTE,
    DO WHILE (INPUT(STAT$51) AND 01H)=0,
        /*  NOTHING  */
        END,
    OUTPUT(CHAR$51)=CHAR,
    END  C$OUT,

C$IN:  PROCEDURE BYTE,
    DO WHILE (INPUT(STAT$51) AND 02H)=0,
        /*  NOTHING  */
        END,
    RETURN INPUT(CHAR$51),
    END  C$IN,
```

Example 2. Simple 8251A Input and Output Routines

The baud timer should be initialized by a code sequence like that shown here. The 80130 logic is actually compatible with the initialization sequence which would be needed to configure timer 2 of an 8253A as a programmable rate generator. The baud rate parameter loaded into the timer is simply the system clock frequency divided by the desired output frequency. No other timers should be affected by user programs.

When the hardware has been initialized, the task calls an operating system procedure called RQ$ RESUME$TASK. This signals the OSP that the task's start-up phase has completed, and that the initialization task (which in this case suspended itself after creating HARD$INIT$TASK) may continue. Since its function is hardware initialization only, HARD$INIT$TASK has no execution phase *per se*. It terminates by calling the procedure RQ$DELETE$TASK, suicidally specifying itself as the task to be deleted.

Exercise 5. Beginners may make two common programming errors when developing OSP tasks. The first is when a task deletes itself without ever resuming the suspended task that created it. The second is to not terminate a task properly, with the result that the processor executes a return instruction when the task's work is done. (However, execution of the task did not originate with a call from the OS.) As with all computers, an OSP will do exactly what it is told. How do you suppose the system would react in each case? (Hint: only one of the two failure modes is predictable.)

You may have noticed three things from this short example and Table 1. First, every OSP call begins with the letters RQ. (PL/M compilers totally ignore dollar signs within symbols; they serve only to split long symbol names to make them easier for humans to read.) The letters RQ don't mean anything in particular; their purpose is to make sure OSP routine names don't conflict with any user symbols. These particular letters were chosen to be compatible with the historical naming convention used by iRMX 86. It may be useful, though, to think of RQ as an abbreviation for REQUEST, implying that the OSP provides useful services at the bidding of application code.

The second thing to notice is that the OSP routine names imply pretty well what each routine does. On the one hand, long procedure names take a little longer to type; on the other, they make code listings much easier to read and understand. In effect, the long names help make OSP code self-documenting. The long names shouldn't hinder code development; rarely can programmers think faster than they can type. If they could, programmer productivity would be measured in thousands of lines per day.

The third thing is that the last parameter in every OSP system call points to a word in which the OSP procedure will return an exception code to the application task. The procedure will return a non-zero exception code in this word if it cannot do its job correctly. This does not always imply that an error occurred; sometimes it just means another task isn't ready to cooperate yet. Sometimes an exception value indicates whether the OSP request was processed immediately or delayed for some reason. In fact, some OSP routines are guaranteed never to return a non-zero exception code, yet the pointer is still required for the sake of consistancy. For a full explanation of the other parameters for the OSP procedures and details on what the different exception codes mean, consult the *iAPX 86/30, 88/30 User's Manual*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To illustrate how the OSP procedures are used, the following code examples implement the machine controller tasks introduced earlier. Appendix B puts all the code examples together, though not in the exact order discussed. *Be Forewarned:* the examples border on trivial. They are in this note to demonstrate how to call system routines with as few lines of code as possible, not to tax the capabilities of the OSP. In fact, none of the tasks even check for exception codes returned by the OSP, under the naive assumption that nothing will go wrong in a debugged program. If you're interested in more elaborate software examples, consult application notes AP-86 and AP-110. These notes focus specifically on iRMX 86, but their methods and much of the code apply equally to the OSP systems.

Simple Time Delays

The STATUS$TASK routine simply monitors eight switches through an input port, and updates eight LEDs with a pattern determined by the switch settings and task status. Specifically, the LEDs display the bitwise Exclusive-OR function of the inputs and an eight-bit software counter maintained by the task. This action will repeat twice per second. The task does nothing between iterations.

The RQ$SLEEP routine gives application tasks a way to release the CPU when it is not needed. Any task calling this routine is "put to sleep" for the amount of time it specifies (from 1 to 65,000 SYSTICK intervals), releasing the CPU to service other tasks in the meantime. After the requested time has transpired, the OSP task will reawaken the task and resume its execution, provided a more important task is not then executing.

The 80130 timer logic generates the fundamental System Tick by dividing the system clock frequency by two, then subdividing that frequency by a 16-bit value specified during the configuration process. The period used here is 5 msec., which would result in an 5 MHz system by dividing the 2.5 MHz internal frequency by 12,500.

> **Exercise 6:** At this rate, what's the longest nap that would result from a single call to RQ$SLEEP? How could this duration be extended?

PL/M listings for the complete STATUS$TASK routine appear in Example 3.

```
STATUS$TASK   PROCEDURE,
    DECLARE STATUS$COUNTER BYTE,
    DECLARE STATUS$EXCEPT$CODE WORD,

    STATUS$COUNTER=0,
    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @STATUS$EXCEPT$CODE),
    DO FOREVER,
        OUTPUT(PPI$B)=INPUT(PPI$A) XOR STATUS$COUNTER,
        STATUS$COUNTER=STATUS$COUNTER+1,
        CALL RQ$SLEEP(100, @STATUS$EXCEPT$CODE),
        END,
    END  STATUS$TASK,
```

Example 3. Status Polling and Reporting Task

Stepper Motor Control

Conceptually, a stepper motor consists of four coils spaced evenly around a rotating permanent magnet. By energizing the coils in various combinations, the magnet can be induced to align itself with the coils, individually or in pairs. A microcomputer can make a stepper motor rotate, step-by-step, in either direction, by emitting appropriate coil control signal patterns at intervals corresponding to the step rate.

The stepper-motor sequencer (Example 4) is an embellished version of STATUS$TASK. The OSP calls are intermixed with a few more statements of application code, and the task uses global variables as delay parameters. The reader may wish to adapt the command interpreter task at the end of this chapter to let the operator modify (read: "play with") these parameters to adjust the motor speed as the program runs.

```
DECLARE CW$STEP$DELAY BYTE,
    CCW$STEP$DELAY BYTE,
    CW$PAUSE$DELAY BYTE,
    CCW$PAUSE$DELAY BYTE,

MOTOR$TASK   PROCEDURE,
    DECLARE MOTOR$EXCEPT$CODE WORD,
    DECLARE MOTOR$POSITION BYTE,
        MOTOR$PHASE BYTE,
    DECLARE PHASE$CODE (4) BYTE
        DATA (00000101B, 00000110B, 00001010B, 00001001B),

    CW$STEP$DELAY=50,     /*INITIAL STEP DELAYS = 1/4 SECOND*/
    CCW$STEP$DELAY=50,
    CW$PAUSE$DELAY=200,   /*PAUSES AFTER ROTATION = 1 SECOND*/
    CCW$PAUSE$DELAY=200,
    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @MOTOR$EXCEPT$CODE),
    DO FOREVER,
        DO MOTOR$POSITION=0 TO 100,
            MOTOR$PHASE=MOTOR$POSITION AND 0003H,
            OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE),
            CALL RQ$SLEEP(CW$STEP$DELAY, @MOTOR$EXCEPT$CODE),
            END,
        CALL RQ$SLEEP(CW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE),
        DO MOTOR$POSITION=0 TO 100,
            MOTOR$PHASE=(100-MOTOR$POSITION) AND 0003H,
            OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE),
            CALL RQ$SLEEP(CCW$STEP$DELAY, @MOTOR$EXCEPT$CODE),
            END,
        CALL RQ$SLEEP(CCW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE),
        END,
    END  MOTOR$TASK,
```

Example 4. Stepper-Motor Controller Task

Real-Time Interrupt Processing

The 80130 supports a two-tiered hierarchy of interrupt processing. The lower-level tier corresponds to the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

traditional concept of hardware interrupt servicing; a routine called an "Interrupt Handler" is invoked by the 80130 internal interrupt control logic for immediate response to asynchronous external events. A short routine like this might, for example, move one character from a USART to a buffer. Interrupt handlers operate with lower-priority interrupts disabled, so it is a good idea to keep these routines as quick as possible.

"Interrupt Tasks," on the other hand, are higher-level tasks which sit idle until "released" by an interrupt handler. The task then executes along with other active tasks, under the control of the OSP. Such a task should be used to perform slower but less time-critical processing when occasions warrant, such as when the aforementioned buffer is full. Moving such additional processing outside the hardware-invoked interrupt handler reduces the worst-case interrupt processing time.

This hierarchy also decreases interrupt latency. Most OSP primitives execute in their own, private "environment" (e.g., with their own stack and data segments) rather than that of the calling task. Interrupt handlers, on the other hand, run in the same environment as the interrupted task. (In fact, the 80130 primitives may themselves be interrupted!) Leaving the CPU segment registers unchanged minimizes software overhead and interrupt response time, but also means that interrupt handlers may not call certain OS routines. An interrupt task, on the other hand, is initiated and suspended by the OSP itself, with no such restrictions.

Let's see how these capabilities would be used. The time delays introduced by the RQ$SLEEP call are only as accurate as the crystal frequency from which they are ultimately derived. This may not be exact enough for critical time-keeping applications, since oscillators vary slightly with temperature and power fluctuation.

To keep track of the time of day, the example system uses a 60-Hz A.C. signal as its time base. (Most power utility companies carefully regulate line frequency to *exactly* 60 Hz, averaged over time.) A signal from the power supply is made TTL-compatible to drive one of the 80130 interrupt request pins. An interrupt handler responds to the interrupts, keeping track of one second's worth of A.C. cycles. An interrupt task counts the seconds by incrementing a series of variables.

Example 5 illustrates the former routine. AC$HANDLER simply increments a variable on each 60-Hz interrupt. Upon reaching 60, it clears the counter and signals TIME$TASK (Example 6).

```
DECLARE AC$CYCLE$COUNT BYTE,

AC$HANDLER   PROCEDURE INTERRUPT 59,     /*VECTOR FOR 80130 INTD*/
    DECLARE AC$EXCEPT$CODE WORD,

    CALL RQ$ENTER$INTERRUPT(AC$INTERRUPT$LEVEL,@AC$EXCEPT$CODE),
    AC$CYCLE$COUNT=AC$CYCLE$COUNT+1,
    IF AC$CYCLE$COUNT >= 60
        THEN DO,
            AC$CYCLE$COUNT=0,
            CALL RQ$SIGNAL$INTERRUPT(AC$INTERRUPT$LEVEL,
                @AC$EXCEPT$CODE),
            END,
        ELSE CALL RQ$EXIT$INTERRUPT(AC$INTERRUPT$LEVEL,
            @AC$EXCEPT$CODE),
    END  AC$HANDLER,
```

Example 5. 60-Hz A.C. Interrupt Handler

In its initialization phase, TIME$TASK sets up the interrupt handler by calling the RQSETINTERRUPT routine. The body of TIME$TASK (the execution phase) is just a series of nested loops counting hours, minutes, and seconds. When TIME$TASK calls RQ$WAIT$INTERRUPT inside its inner-most loop, the OSP suspends execution of the task until AC$HANDLER signals that another second's worth of A.C. cycles has elapsed. Thus, interrupt handlers can serve to "pace" interrupt tasks. After a day, TIME$TASK completes and deletes itself.

```
DECLARE SECOND$COUNT BYTE,
    MINUTE$COUNT BYTE,
    HOUR$COUNT BYTE,

TIME$TASK   PROCEDURE,
    DECLARE TIME$EXCEPT$CODE WORD,

    AC$CYCLE$COUNT=0,
    CALL RQ$SET$INTERRUPT(AC$INTERRUPT$LEVEL,01H,
        INTERRUPT$PTR(AC$HANDLER),DATA$SEG$ADDR BASE,
        @TIME$EXCEPT$CODE),
    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@TIME$EXCEPT$CODE),
    DO HOUR$COUNT=0 TO 23,
        DO MINUTE$COUNT=0 TO 59,
            DO SECOND$COUNT=0 TO 59,
                CALL RQ$WAIT$INTERRUPT(AC$INTERRUPT$LEVEL,
                    @TIME$EXCEPT$CODE),
                IF SECOND$COUNT MOD 5 = 0
                    THEN CALL PROTECTED$CRT$OUT(BEL),
                END,    /* SECOND LOOP */
            END,    /* MINUTE LOOP */
        END,    /* HOUR LOOP */
    CALL RQ$RESET$INTERRUPT(AC$INTERRUPT$LEVEL,
        @TIME$EXCEPT$CODE),
    CALL RQ$DELETE$TASK(0,@TIME$EXCEPT$CODE),
    END  TIME$TASK,
```

Example 6. Interrupt Task to Maintain Time of Day

Exercise 7: The time maintained by TIME$TASK is consistently wrong, unless the system resets at midnight. Aside from that, how much error would accumulate per month had TIME$TASK paced its inner loop by calling RQ$SLEEP if the system oscillator was 00.01% off? How does this compare with a cheap digital watch? How much error will accumulate from the 60-Hz time base described?

TIME$TASK incorporates another gimmick: every five seconds it sends an ASCII "BEL" character (07H) to the console to make it beep, by calling a routine called PROTECTED$OUTPUT. This lead-in gives us a chance to discuss OSP provisions for task synchronization and mutual exclusion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Mutual Exclusion

Whenever system resources (e.g., the console) are shared among multiple concurrent tasks, the software designer must be aware of the potential for conflicts. In single-threaded (as opposed to multitasking) programs, the easiest way to transmit characters is by calling a console output routine (written by the user or supplied by the OS) which outputs the character code. (Remember the examples following the hardware initialization routine?)

This approach presents two problems in a multitasking system. One is efficiency: a high-priority task could hang up the whole system while it waits for a printer solenoid to energize, induce a magnetic field, accelerate the hammer, contact a daisy-wheel spoke, move it up to the ribbon, and press them both against the paper. This waste of time is termed "busy waiting," and should always be avoided. By OSP standards, even 1/30 of a second can seem interminable; if the printer is otherwise occupied, the whole system could shut down indefinitely.

Aside from efficiency, though, there is a more serious synchronization problem here. Assume Task A has a higher priority than Task B. Task A is asleep. Task B calls a subroutine to poll the USART and transmit a character. The USART becomes ready. When this is detected, the subroutine prepares to output the character to the USART . . . .

Time out! Task A just woke up and starts running. Task A wants to transmit its own character. It calls its own output routine, checks the USART, finds it available, sends it a new character, and goes back to sleep (or suspends itself, or awaits another interrupt—whatever).

Now Task B continues. It "knows" the USART is available, having dutifully monitored it earlier. Task B's character goes out to the USART. The USART goes out to lunch. (In practice, the USART will probably just transmit corrupted data; still, its operating requirements have been violated.)

In Task B's output routine, the sequence of statements from when the peripheral is found to be ready to when the next character is written constitutes a "critical region" (a.k.a. "critical section" or "non-interruptable sequence"). Recognizing such regions and handling them correctly is an important concern in any multitasking system, so the OSP provides several facilities —interrupt control, regions and mailboxes—to help handle general synchronization and mutual exclusion problems. Which one to choose depends on the circumstance.

Exercise 8: In this example, would it be better if Tasks A and B shared a single output routine, so that only one section of code sent data to the USART? Convince yourself that the same (or worse!) problems could still arise.

Sometimes critical sections can be protected by just disabling interrupts at appropriate points in the application software. To maintain the integrity of an iAPX 86/30 system, *application code must never execute the STI, CLI, or HLT instructions* (ENABLE, DISABLE, or HALT statements in PL/M), nor can it access the interrupt control logic directly. Instead, the interrupt status should be controlled with the OSP RQ$ENABLE and RQ$DISABLE procedures; routines should be halted via RQ$SUSPEND or RQ$WAIT$INTERRUPT.

Back to TIME$TASK: we want to transmit BELs to the console every five seconds. The console output task will be transmitting other characters. A "clever" programmer may recognize that this will lead to a critical section and analyze the situation as follows:

1. A hazard would arise if TIME$TASK sends out a beep when CONSOLEOUTTASK is using the USART;
2. TIME$TASK will only execute after being signaled by ACHANDLER;
3. ACHANDLER only reponds to an external interrupt.

"Therefore, all CONSOLEOUTTASK has to do to be safe is disable the 60-Hz interrupt around its output routine."

Not quite. There are still potential hazards. Suppose CRTOUTTASK has the same priority as TIME$TASK. TIME$TASK may already have been signaled by ACHANDLER and be ready to run when CRTOUTTASK completes. An otherwise unrelated event—another interrupt, for instance—could momentarily suspend CRTOUTTASK during the critical region with A.C. interrupts disabled. When the OSP returns to that level, it might resume with TIME$TASK, *not* CRT$OUT$TASK. This could lead to the same malfunctions as before, so disabling 60-Hz interrupts didn't help. This series of worst-case assumptions is admittedly convoluted, but the resulting sporadic errors are among the hardest of all bugs to squash.

The problem is that this attempted solution involves too much interaction between tasks, making it confusing and error-prone. Even if some scheme of priority-level assignments and task interactions could be made to work, later modifications or simple additions to the job

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

could cause bugs to reappear. (The analogy of an unexploded time bomb comes to mind.)

A simpler solution would be one corresponding more closely with the problem. Accordingly, the OSP supports several primitives just to supervise and control access to critical regions.

One of the OSP "data types" is a data structure called a "Region," which can be used by application code to control access to a shared port or some other resource. A task wishing access to the resource should call the OSP procedure RQ$RECEIVE$CONTROL before trying to access that resource; when done it must call RQ$SEND$CONTROL.

The OSP keeps track of which regions are in use. As long as a region is busy (i.e., has been entered but not yet exited), the OSP will prevent other tasks from entering the region by putting them to sleep. The OSP keeps a queue of all tasks waiting for the busy region. When the region later becomes available (i.e., when the task controlling the region calls RQ$SEND$CONTROL), one of the sleeping tasks—either the highest priority or the most patient—will be awakened, granted control of the region, and sent on its way. (When a region is created, the OSP is told whether to awaken tasks waiting for the region based on their priority or how long they have been waiting.) Effectively, a call to RQ$RECEIVE$CONTROL will not return to the application task until the resource in question becomes available.

The PROTECTEDCRTOUTPUT (Example 7) demonstrates this protocol. The routine is declared reentrant which means (by definition) the routine may be interrupted and restarted safely. A reentrant routine may be shared by a number of tasks, instead of replicating the same code throughout the application.

```
PROTECTED$CRT$OUT:   PROCEDURE (CHAR) REENTRANT;
    DECLARE CHAR BYTE;
    DECLARE CRT$EXCEPT$CODE WORD;
    CALL RQ$RECEIVE$CONTROL(CRT$REGION$TOKEN,@CRT$EXCEPT$CODE);
    DO WHILE (INPUT(STAT$51) AND 01H)=0;
        /*  NOTHING  */
        END;
    OUTPUT(CHAR$51)=CHAR;
    CALL RQ$SEND$CONTROL(@CRT$EXCEPT$CODE);
    END  PROTECTED$CRT$OUT;
```

Example 7. CRT Output Routine Protected by Region Protocol

As a concession to simplicity, PROTECTEDCRTOUTPUT does use a form of the busy waiting method described earlier. The maximum delay at 9600 baud is only one millisecond, however, much shorter than a system tick. Besides, tasks performing character I/O will all have low priority levels, so the OSP would just delay them if anything more urgent comes up.

> **Exercise 9:** Decide whether this explanation is a feeble attempt at rationalization, or a well-justified engineering trade-off.

Inter-Task Communication

But what if a high priority task must output a string of characters, or the peripheral response time is too long? Busy-waiting may not be acceptable. Alternatively, the output routine could buffer the data and service the USART within an interrupt routine. Another would be to simply pass the data off to a special (low-priority) output task and continue.

Tasks pass information to each other via something called a "message." A message may be the token for any type of OSP object, but the most common and most flexible type is called a "memory segment." In our example, segments will be used to carry strings of ASCII characters between tasks, so we'll examine segments first. Message formats are defined by the individual application programmer—make sure the sending and receiving tasks assume the same format!

A memory segment is just a section of contiguous-system RAM allocated (set aside) by the OSP at the request of an executing task. The OSP keeps track of a free memory "pool," which is initially all unused RAM in the system. When a task needs some RAM, it tells the RQ$CREATE$SEGMENT procedure how much it wants. The OSP finds a suitable memory block in the pool, and returns a 16-bit token defining its location. (If not enough memory is available, the procedure returns an exception code.)

The token is the base portion of pointer to the first usable byte of the segment, with the offset portion assumed to be zero. (The token values for all other objects have no physical significance.) Knowing this, it's possible to access elements of the segment as the application warrants.

The subroutine in Example 8 shows how to request a segment and construct a message. PRINT$TIME sends the ASCII values of the time-of-day counters (maintained in TIME$TASK) to the CRT output task described later. The message format adopted for these examples will consist of a byte giving the message

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

length, followed by that number of ASCII characters. Figure 10 shows this format.

```
PRINT$TOD:  PROCEDURE;
    DECLARE TOD$MESSAGE$TOKEN WORD;
    DECLARE TOD$EXCEPT$CODE WORD;
    DECLARE TOD$SEGMENT$OFFSET WORD,
        TOD$SEGMENT$BASE WORD;
    DECLARE TOD$SEGMENT$PNTR POINTER AT (@TOD$SEGMENT$OFFSET);
    DECLARE TOD$TEMPLATE (28) BYTE
        DATA (27, 'THE TIME IS NOW hh:mm:ss.', CR, LF);
    DECLARE TOD$STRING BASED TOD$SEGMENT$PNTR (28) BYTE;
    DECLARE TOD$STRING$INDEX BYTE;

    TOD$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(28, @TOD$EXCEPT$CODE);
    TOD$SEGMENT$BASE=TOD$MESSAGE$TOKEN;
    TOD$SEGMENT$OFFSET=0;
    DO TOD$STRING$INDEX=0 TO 27;
        TOD$STRING(TOD$STRING$INDEX)=
            TOD$TEMPLATE(TOD$STRING$INDEX);
        END;
    TOD$STRING(17)=ASCII$CODE(HOUR$COUNT/10);
    TOD$STRING(18)=ASCII$CODE(HOUR$COUNT MOD 10);
    TOD$STRING(20)=ASCII$CODE(MINUTE$COUNT/10);
    TOD$STRING(21)=ASCII$CODE(MINUTE$COUNT MOD 10);
    TOD$STRING(23)=ASCII$CODE(SECOND$COUNT/10);
    TOD$STRING(24)=ASCII$CODE(SECOND$COUNT MOD 10);
    CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
        TOD$MESSAGE$TOKEN, 0, @TOD$EXCEPT$CODE);
    RETURN;
    END  PRINT$TOD;
```

Example 8. Subroutine to Send Time-of-Day Message to Output Task

We're coding PRINT$TIME here (see Example 8), while TIME$TASK is fresh in our minds. It will actually be called by (and is therefore considered a part of) KEYBOARD$TASK. Note that while tasks are written as individual procedures, they need not be fully self-contained: outside procedures should be used to help organize and structure the code.

The first thing PRINT$TIME does is have the OSP create a segment of suitable length, and copies a "message template" into the segment, byte by byte. Then it converts the TIME$TASK counter values to ASCII, filling in blanks in the template. Finally, it sends the token for the message to the CRT mailbox.

To repeat, these examples are intended to illustrate use of the OSP routines assuming minimum familiarity with PL/M. Better programming practices might take advantage of PL/M literals, structures and the array LENGTH function to build the message, rather than the inflexible constants shown here. Some of these techniques are suggested by PRINT$STATUS (Example 9), which indicates the binary status of the input switches.

```
PRINT$STATUS:  PROCEDURE;
    DECLARE STATUS$MESSAGE$TOKEN WORD;
    DECLARE STATUS$EXCEPT$CODE WORD;
    DECLARE STATUS$SEGMENT$OFFSET WORD,
        STATUS$SEGMENT$BASE WORD;
    DECLARE STATUS$SEGMENT$PNTR POINTER
        AT (@STATUS$SEGMENT$OFFSET);
    DECLARE STATUS$TEMPLATE (40) BYTE DATA
        (39, 'THE SWITCHES ARE NOW SET TO        B', CR, LF);
    DECLARE STATUS$STRING BASED STATUS$SEGMENT$PNTR (40) BYTE;
    DECLARE STATUS$STRING$INDEX BYTE;
    DECLARE BIT$PATTERN BYTE;

    STATUS$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(40,
        @STATUS$EXCEPT$CODE);
    STATUS$SEGMENT$BASE=STATUS$MESSAGE$TOKEN;
    STATUS$SEGMENT$OFFSET=0;
    DO STATUS$STRING$INDEX=0 TO 39;
        STATUS$STRING(STATUS$STRING$INDEX)=
            STATUS$TEMPLATE(STATUS$STRING$INDEX);
        END;
    BIT$PATTERN=INPUT(PPI$A);
    DO STATUS$STRING$INDEX=29 TO 36;
        STATUS$STRING(STATUS$STRING$INDEX)=
            ASCII$CODE(BIT$PATTERN AND 01H);
        BIT$PATTERN=ROR(BIT$PATTERN, 1);
        END;
    CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
        STATUS$MESSAGE$TOKEN, 0, @STATUS$EXCEPT$CODE);
    END  PRINT$STATUS;
```

Example 9. Subroutine to Send Status Report Message to Output Task

Exercise 10: One input port is read by both STATUS$TASK and PRINT$STATUS. Does this constitute a shared resource? A critical region?

Exercise 11: PRINT$TIME reads the counts maintained by TIME$TASK, but doesn't alter them. Forced mutual exclusion is generally mandatory when multiple tasks perform read/modify/write sequences on a given variable. Can PRINT$TIME make TIME$TASK malfunction? What about the opposite case? If this failure mode was deemed unacceptable, how could it be protected?

Mailboxes

The data in a message doesn't actually move or get copied from source to destination when the message is sent; this would be too slow with long messages. Rather, the OSP "carries" the message's token from task to task via a data structure cleverly termed a mailbox. If one task must send messages to another, a mailbox must be created to hold them. The sender calls the RQ$SEND$MESSAGE to put a message token into the mailbox. If the receiver isn't ready for the message yet, the OSP puts the message token into an ordered queue. When the receiver calls RQ$



Figure 10. Message Formats Expected by Output Task

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RECEIVE$MESSAGE later, the OSP will give it the tokens one at a time.

What happens if a task tries to receive a message when the mailbox is empty? (This is quite possible, since tasks do run asynchronously.) What token would the OSP return?

In the simple case . . . it doesn't! Instead of returning right away with no data, the OSP will wait until data *is* available. In the meantime, the OSP puts the receiving task to sleep, remembering that it is waiting for a message at that mailbox. The next time a message is sent to that mailbox, the OSP will awaken the receiving task, give it the token, and—if its priority is high enough—resume its execution. Alternatively, receiving tasks may elect to not wait if the mailbox is empty, or to wait only a specified time.

Many tasks may actually send and receive messages through a single mailbox, with messages being queued in the order that the RQ$SEND$MESSAGE calls are executed. The OSP also maintains a list of tasks waiting to receive messages from an empty mailbox, analogous to the queued tasks waiting for region control. As each message is sent to the mailbox, it is passed immediately to a waiting task, either the one waiting the longest or the one with the highest priority (likewise determined by a parameter specified when the mailbox is created).

> **Exercise 12:** Under what conditions could a mailbox's message queue contain messages waiting to be received, while the task queue contains tasks waiting for messages? Ignore the possibility that this may happen momentarily during the implementation of either routine. If you think of any such circumstances, please contact the author.

Example 10 shows a task which prints the messages sent above. Upon receiving a message token, CRTOUTTASK determines the message length from the first two bytes, and sequentially prints each element of the string through the PROTECTEDCRT OUTPUT routine explained earlier. When done, the segment containing the message is deleted, returning its RAM to the free-memory pool.

A few words are in order about the segment accessing techniques demonstrated here. PL/M-86 has a special data type, called a "pointer," used to indirectly access other PL/M variables. OSP application programs must be compiled with the "compact" or "large" model specified. This tells the compiler to implement pointers as 32-bit double words corresponding to the two parts (base:offset) of the 8086 machine-segmented addressing scheme. PL/M-86 tries to shield the programmer

```
CRT$OUT$TASK   PROCEDURE;
    DECLARE MESSAGE$LENGTH BYTE,
    DECLARE MESSAGE$TOKEN WORD,
    DECLARE RESPONSE$TOKEN WORD,
    DECLARE MESSAGE$EXCEPT$CODE WORD,
    DECLARE MESSAGE$SEGMENT$OFFSET WORD,
        MESSAGE$SEGMENT$BASE WORD;
    DECLARE MESSAGE$SEGMENT$PNTR POINTER AT
        (@MESSAGE$SEGMENT$OFFSET),
    DECLARE MESSAGE$STRING$CHAR BASED MESSAGE$SEGMENT$PNTR BYTE,

    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @MESSAGE$EXCEPT$CODE),
    DO FOREVER;
        MESSAGE$TOKEN=RQ$RECEIVE$MESSAGE(CRT$MAILBOX$TOKEN, OFFFFH,
            @RESPONSE$TOKEN, @MESSAGE$EXCEPT$CODE),
        MESSAGE$SEGMENT$OFFSET=0;
        MESSAGE$SEGMENT$BASE=MESSAGE$TOKEN,
        MESSAGE$LENGTH=MESSAGE$STRING$CHAR,
        DO MESSAGE$SEGMENT$OFFSET=1 TO MESSAGE$LENGTH,
            CALL PROTECTED$CRT$OUT(MESSAGE$STRING$CHAR),
            END;
        CALL RQ$DELETE$SEGMENT(MESSAGE$TOKEN, @MESSAGE$EXCEPT$CODE),
        END;    /* OF FOREVER-LOOP */
END  CRT$OUT$TASK;
```

Example 10. Task to Transmit Messages to the CRT

from the details, yet at times the two parts must be manipulated separately (for instance, to access data in an OSP segment knowing only the segment token/base value).

To get around this, these examples assign a pair of word variables to the same address as a PL/M pointer variable. Each representation is then an alias for the other. To determine the base or offset value of an item of data, load the pointer variable with a pointer to the item and then reference the appropriate field of the overlayed pair of word variables. To "build" an arbitrary pointer, assign computed values to the base and offset fields and then access the data item via the composite pointer.

> **Exercise 13:** PL/M 86 does not have built-in functions to separate the high and low-order words of a pointer variable. Does this seem to be a weakness in the language? Bear in mind that the machine representation for pointers varies depending on which programming model is specified at compilation time. When the "small" model is selected, the compilers take advantage of a 16-bit pointer representation for faster and more compact code.

Console Command Interpreter

If a system has a console keyboard, it's probably used to accept and interpret operator commands. For this demonstration system, the lowest priority of all tasks is a simple-minded routine which polls the USART until a character has been received, and immediately echoes it by calling—you guessed it!—PROTECTED CRTOUTPUT. Thus, the keyboard is "alive"; it responds immediately to keystrokes, so the operator can type whatever nonsense he desires while everything else is going on.

Ten of the keys (digits 0 through 9), invoke special commands which illustrate interactions between the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

multiple tasks. Commands 0 and 1 print out the time and status messages; the rest suspend and resume various tasks, as shown by Table 2. The code for COMMAND$TASK appears in Example 11.

Initialization Task

Now that the application tasks have been written, we can write the initialization task.

All applications require a special type of task to initialize system variables and peripherals and create tasks and other objects used by the application. It, too, is written as a PL/M procedure, and can thus be divided conceptually into the same three phases.

Example 12 shows such a task for the demonstration system. The first thing INIT$TASK does is determine the base address of the job data segment by assigning pointer DATASEGPTR with its own address. Next it calls the RQGETTASK$TOKENS routine, which tells the task what token value the OSP assigned it at run time. It then initializes the system peripherals by creating the hardware initialization task discussed above; this code could have been integrated into INIT$TASK itself just as easily. During its own "execution" phase, INIT$TASK calls routines to create the OSP data structures shared by the application tasks: the REGION controlling access to the USART, and the MAILBOX repository for output messages. INIT$TASK creates the application tasks themselves by calling RQ$CREATE$TASK.

Though not always required, it is common practice for the overall initialization task to suspend itself after creating each offspring, to let the newborn task get started. Under this convention, each offspring task must resume the initialization task by calling the

```
COMMAND$TASK   PROCEDURE.
    DECLARE CONSOLE$CHAR BYTE.
    DECLARE COMMAND$EXCEPT$CODE WORD.

    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @COMMAND$EXCEPT$CODE).
    DO FOREVER.
        CONSOLE$CHAR=0$IN AND 7FH.
        CALL PROTECTED$CRT$OUT(CONSOLE$CHAR).
        IF CONSOLE$CHAR=CR
            THEN CALL PROTECTED$CRT$OUT(LF);
        IF (CONSOLE$CHAR >= '0') AND (CONSOLE$CHAR <= '9')
            THEN DO.
                CALL PROTECTED$CRT$OUT(CR).
                CALL PROTECTED$CRT$OUT(LF).
                DO CASE (CONSOLE$CHAR-'0').
                    CALL PRINT$TOD.
                    CALL PRINT$STATUS.
                    CALL RQ$SUSPEND$TASK(CRT$OUT$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$RESUME$TASK(CRT$OUT$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$DISABLE!AC$INTERRUPT$LEVEL.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$ENABLE(AC$INTERRUPT$LEVEL.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$SUSPEND$TASK(MOTOR$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$RESUME$TASK(MOTOR$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$SUSPEND$TASK(STATUS$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    CALL RQ$RESUME$TASK(STATUS$TASK$TOKEN.
                        @COMMAND$EXCEPT$CODE).
                    END.    /* OF CASE-LIST */
            END.    /* OF COMMAND PROCESSING */
    END.
END COMMAND$TASK.
```

Example 11. Task to Accept and Process Keyboard Commands

```
INIT$TASK  PROCEDURE PUBLIC.
    DECLARE INIT$EXCEPT$CODE WORD.

    DATA$SEG$PTR=@INIT$TASK$TOKEN.  /*LOAD DATA SEGMENT BASE*/
    CRT$MAILBOX$TOKEN=RQ$CREATE$MAILBOX(0,@INIT$EXCEPT$CODE).
    CRT$REGION$TOKEN=RQ$CREATE$REGION(0,@INIT$EXCEPT$CODE).
    INIT$TASK$TOKEN=RQ$GET$TASK$TOKENS(0,@INIT$EXCEPT$CODE).
    HARDWARE$INIT$TASK$TOKEN=RQ$CREATE$TASK
        (110,@HARDWARE$INIT$TASK,DATA$SEG$ADDR BASE,0,300,
        0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    STATUS$TASK$TOKEN=RQ$CREATE$TASK(110,@STATUS$TASK,
        DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    MOTOR$TASK$TOKEN=RQ$CREATE$TASK(110,@MOTOR$TASK,
        DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    TIME$TASK$TOKEN=RQ$CREATE$TASK(120,@TIME$TASK,
        DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    CRT$OUT$TASK$TOKEN=RQ$CREATE$TASK(120,@CRT$OUT$TASK,
        DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    COMMAND$TASK$TOKEN=RQ$CREATE$TASK(130,@COMMAND$TASK,
        DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE).
    CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE).
    CALL RQ$END$INIT$TASK.
    CALL RQ$DELETE$TASK(0,@INIT$EXCEPT$CODE).
    END  INIT$TASK.
```

Example 12. Task to Initialize System Software

Table 2. Special Console Commands

Key	Function
0	Send Time-of-day message to CRT.
1	Send status update message to CRT.
2	Suspend CRT output task. The OSP will automatically save messages to the task in the CRT mailbox queue.
3	Resume CRT output task. Queued messages will be displayed.
4	Disable 60-Hz interrupt-driven time base. Time-of-day clock will stop.
5	Enable 60-Hz time base to resume clock execution.
6	Suspend motor control task. Motor will stop.
7	Resume motor control task. Note that if task was suspended 17 times, it must be resumed 17 times.
8	Suspend status polling task. Lights indicating system status will freeze in current state.
9	Resume status polling task.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RQ$RESUME$TASK routine when its own local initialization is complete. This convention is called synchronous initialization; its purpose is to ensure that each task is allowed to complete its own start-up phase before the next task is created. Otherwise, there's a risk that higher-priority tasks created later could start executing before earlier tasks were ready for them, with (at best) unpredicatable results.

When all the tasks have been created, INIT$TASK has served its purpose. It must then call RQ$SEND$INIT$TASK. This short procedure (actually self-contained in an OSP Support Package interface library, not built into the 80130) tells the OSP that all the offspring tasks have been created for a given job. At this point, INIT$TASK could continue with non-initialization activities. The code for KEYBOARD$TASK might have been implemented here, for example. Since this example has nothing more to do, INIT$TASK deletes itself with a final call to RQ$DELETE$TASK.

Code Translation

That's all, folks. Mix together the above code fragments, declare literals and global variables, and compile until done (about four minutes). The source file name selected for this example is AP130.PLM. The compiler will produce two files: an annotated source listing (named AP130.LST) reproduced *in toto* in Appendix B, and a relocatable object file (AP130.OBJ) which will be used in the installation procedure discussed next.

High-Level Parameter Passing Conventions

Well-designed programs generally rely on subprograms ("procedures" in PL/M terminology) for often-repeated instruction sequences, or to perform machine-level operations within High-Level Language programs. PL/M-86 and other Intel high-level languages use a standard set of conventions to pass parameters and results between procedures; assembly language programmers are advised to adhere to these conventions for software compatibility.

Before calling a subroutine or function, input parameters must be pushed sequentially onto the stack, in the order (left-to-right) they appear in the procedure parameter list. When eight-bit parameters are pushed, the high-order byte associated with them is undefined. Thirty-two-bit pointer values are pushed in two steps, offset word before base word. The stack "grows" down, so the left-most parameter will have highest-numbered address.

Functions which return a byte or word value (i.e., typed procedures) do so in the CPU AL or AX registers. Pointers are returned through the ES:AX register pair. The *PL/M Programming Manual* explains these conventions more fully.

One way to see how an assembly language routine would interface with PL/M is to first write a dummy PL/M procedure using the same parameter sequence as the desired assembly language routine. Compile this procedure with the compiler CODE switch set. The listing will then include the appropriate assembly language instruction sequence, and may be followed as a pattern for the final routine.

SOFTWARE CONFIGURATIONS & INTEGRATION

When the application code has been written and compiled, the hardest part of program development is over. Before the code may be executed, though, the OSP must be told various things about the system hardware environment, desired software options, application job characteristics, and so forth.

This information is conveyed during a multi-phase sequence of steps collectively called the Configuration process. Though the process is somewhat lengthy and time-consuming, it is also very "mechanical"; the person doing the work does not need to understand any of the application code or even know what it does. Normally, configuration would be performed by a technician or a single member of the programming team, aided by appropriate SUBMIT command files. This chapter shows the full configuration and installation process for the demonstration system. For more details, refer to the *OSP User's Manual*.

The three phases of the configuration are:

1. Generating, linking, and locating OSP support code required for the EPROM immediately above the 80130 address space;
2. Linking and locating the object file for the application job developed in Section IV;
3. Creating, linking, and locating a short module (called the Root Job) which initializes the OSP and application jobs when system is reset.

Finally, of course, the absolute code resulting from each phase must be programmed into EPROMs or loaded into a test system before it can be executed.

Before starting, though, it is beneficial to draw up a memory map for host system hardware, to determine what sections of memory are available. This map will be filled in as each module is linked and located.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The prototype system memory space has two areas of interest: addresses 00000H through 01FFFH contain RAM, while 0FC000H through 0FFFFFH contain EPROM. Since the CPU uses the first 1K bytes of RAM for the CPU interrupt pointers, and the last 16 bytes for the restart sequence, these areas should be recorded on the map. For reference purposes, Figure 11 also indicates that addresses 0F8000H through 0FBFFFH enable the 80130 firmware. All this is shown in Figure 11.

Generating the OSP Support Code

The OSP support code "customizes" the OSP firmware for a particular hardware environment, initializes the system, and supports extended software capabilities.

To define the hardware environment, the user creates a source file which invokes a series of Intel-supplied macros. Parameters for these macros specify the 80130 I/O base address, SYSTICK interval (in system clock cycles), and how the interrupt request pins will be used.

For instance, the code example in Figure 12 defines the prototype system hardware. This source file must be assembled, linked with several libraries from the OSP support disk, and located to produce the actual OSP support code. Figure 13 shows the actual sequence of commands needed. The DATA starting address specified within the LOC86 parameter list (00400H) is the first free byte of system RAM (see Figure 11); the CODE address (0F8000H) is simply the 80130 firmware starting address.

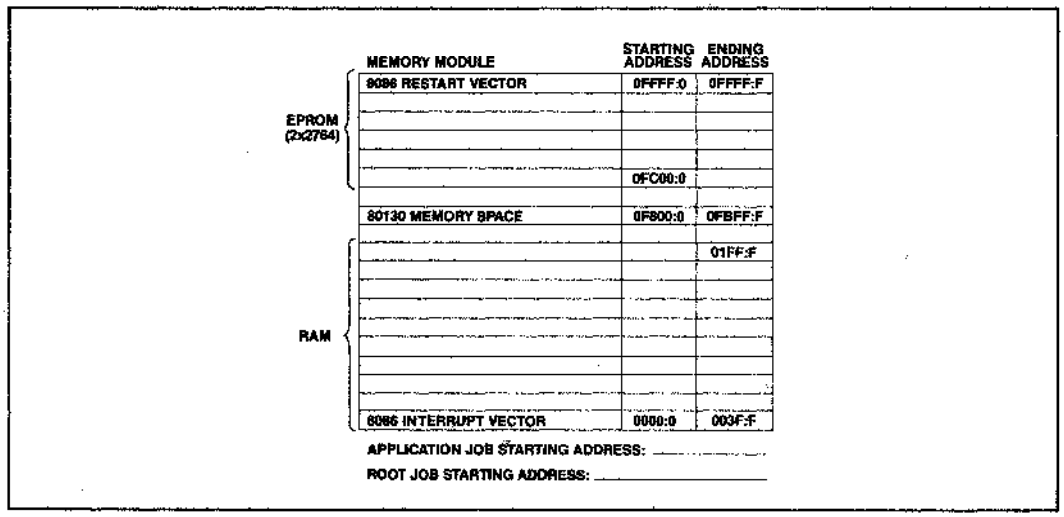


Figure 11. Example System Memory Map

```
$TITLE(80130 DEVICE CONFIGURATION TABLE)
NAME DEVCF

$INCLUDE( F1 NDEVCF MAC)

%MASTER_PIC(80130, 2000H, 0, 0)

, SLAVE_PIC( SLAVE_TYPE, BASE_PORT, EDGE_VS_LEVEL, MASTER_LEVEL )

%TIMER(80130, 200BH, 28H, 12500)

, NDP_SUPPORT( ENCODED_LEVEL )

END
```

Figure 12. 80130 Device Configuration Table

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
 F0 ASM86  F1 SUP130 A86 PRINT( F1 SUP130 LST) ERRORPRINT &
MACRO(80) PAGEWIDTH(132)
,
 F0 LINK86 &
 F1 OSX LIB(OSXB6,OSXCNF),  &
 F1 NUC1 LIB(NBEGIN),       &
 F1 ODEVCF OBJ,             &
 F1 OSX LIB,                &
 F1 NUC1 LIB,               &
 F1 OSX LIB,                &
 F1 NUC2 LIB,               &
 F1 OSX LIB,                &
 F1 NUC4 LIB,               &
 F1 OSX LIB,                &
 F1 NURSLV LIB,             &
 F1 OSX LIB                 &
TO  F1 SUP130 LNK MAP PRINT( F1 SUP130 MP1) NAME(MINIMAL_B0130)
,
 F0 LOC86                                          &
 F1 SUP130 LNK TO  F1 SUP130 MAP PRINT( F1 SUP130 MP2) SC(3) &
SEGSIZE(STACK(0))                                  &
ADDRESSES(CLASSES(CODE(0F800H),DATA(00400H)))          &
ORDER(CLASSES(DATA,STACK))                         &
OBJECTCONTROLS(NOLINES,NOCOMMENTS,NOSYMBOLS)
,
```

Figure 13. Support Code Configuration Commands

A reliable and relatively straightforward way to perform this step is to create a file containing the exact command sequence shown in Figure 13 and execute this file using the SUBMIT utility program. Of course, the example assumes SUBMIT, ASM86, LINK86, and LOC86 are all on drive :F0:, and that the various libraries have been copied from the support disk to drive :F1:.

(An alternate, support-code configuration scheme lets the user modify the OSP software characteristics in special situations. A programmer working with iRMX 86, for instance, may wish to augment the OSP firmware to support all the iRMX Nucleus primitives. This would be done by editing and assembling file 0TABLE.A86 to select from a menu of software options, and modifying the linkage step slightly to include one of the iRMX 86 libraries. The OSP built-in features are more than sufficient for the purposes of this note, though, so only the first approach is illustrated.)

Appendix D reproduces the Locate map file produced during this phase. Near the end of file SUP130.MP2 is a table of memory usage, showing that the last bytes of RAM and ROM consumed are 00A6: FH and 0FC61: FH, respectively. Update Figure 11 with this information. (The final version of the demonstration-system memory map appears in Appendix C.) This phase needn't be repeated unless the system hardware characteristics change.

Application Code Configuration

After compiling the application job, it must be linked with a library of interface routines from the support diskette, and located within available memory. Use RPIFC.LIB or RPIFL.LIB, depending on whether the job was compiled with the Compact or Large software model. Figure 14 is a command sequence file suggested for this purpose. Again, the starting addresses specified for LOC86 are taken from the system memory map.

Whenever the support code is reconfigured, check SUP130.MP2 to see if its memory needs have changed. If so, the application-job-configuration command file will need to be edited. This is still a lot simpler (not to mention more reliable) than retyping the whole sequence each time application jobs are revised. Readers familiar with the capabilities of the SUBMIT program may prefer to represent these variables by parameters, such that they may be easily specified each time the command file is invoked.

As in the first phase, examine the locate map ("AP130.MP2", reproduced in Appendix E) after the application code has been configured and update the memory map. Also, note the segment and offset values assigned to the initialization task. These will be needed later.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating the Root Job

By now, all of the code needed to execute the application program has been prepared and is ready to run —except it has no way to get it started! The OSP hardware and system data structures must be initialized before INIT$TASK can be created. A short module called the Root Job performs this function.

The process closely resembles the one which produced the OSP support code. First, determine various system characteristics. Then create a file defining these characteristics as macro input parameters. Finally, assemble, link, and locate the file to produce the final code.

Figure 15 is the Root Job source file for the demonstration system, dubbed RJB130.A86. It consists of just five macro calls. The %JOB macro defines certain characteristics of the application job; for a full description see the *OSP User's Manual*. One of these parameters is the initialization-task starting address (noted in the last step), which will likely change with each iteration of the application software.

The two %SAB macros define "System Address Blocks"—sections of the overall memory space which the OSP should not consider "free space." Note that the first invocation blocks off the RAM addresses consumed so far in the memory map, plus an extra 140H bytes reserved for the Root Job initialization stack.

```
;
;   SUBMIT FILE TO LINK APPLICATION JOB TO INTERFACE LIBRARY
;   AND LOCATE RESULTING OUTPUT.
;   REVISED 10/23/81 - JHW
;
LINK86 :F1:AP130.OBJ, :F1:RPIFC.LIB TO :F1:AP130.LNK        &
    MAP PRINT(:F1:AP130.MP1)

LOC86 :F1:AP130.LNK TO :F1:AP130                             &
    ORDER (CLASSES(DATA,STACK,MEMORY))                       &
    SEGSIZE (STACK (0))                                      &
    ADDRESSES (CLASSES (DATA (00A7OH),                       &
                        CODE (0FC620H)))                     &
    MAP PRINT (:F1:AP130.MP2)                                &
    OBJECTCONTROLS (NOLINES,NOCOMMENTS,NOPUBLICS,NOSYMBOLS)

OH86 :F1:AP130.TO :F1:AP130.H86

COPY :F1:AP130.MP1 TO LP

COPY :F1:AP130.MP2 TO LP
```

Figure 14. Job Configuration Commands

```
;
; SOURCE PROGRAM DEFINING CHARACTERISTICS OF ROOT JOB FOR
; AP-130 DEMONSTRATION PROGRAM  (JHW - 10/29/81)
;
$INCLUDE(:F1:CTABLE.MAC)
;
%SAB(0,00C0,U)
%SAB(0200,FFFF,U)
%JOB(0,0C0H,100H,0FFFFH,0FFFFH,1,0,0,1,0,100,0FC62:06B5,0,0,0,200H,0)
%DSX    (0F8000H,N)
%SYSTEM(F800,0,4,N,N,1)
;
END
```

Figure 15. Root Job Configuration File

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(After completing this phase, examine RJB130.MP2 to confirm that 140H is the correct number.) The second %SAB invocation excludes addresses 02000H through 0FFFFFH, all of which is non-RAM, either EPROM, 80130 firmware, or non-existent. The %SYSTEM macro defines system-wide software parameters.

Figure 16 is a command file to translate, link, and locate the root job. Once again, the LOC86 parameters come from Figure 11. The listings produced during this phase are reproduced in Appendix F. The final memory map appears in Appendix C.

EPROM Programming

We are now ready to program EPROMs with the program modules linked and located above. Intel's Universal PROM Programmer (UPP) and a control program called the Universal Prom Mapper (UPM) will be used in this step. Particular commands to the UPM will vary with program size, memory location, and EPROM type, but the general sequence should resemble that shown here.

The first step is to invoke UPM and initialize the programming system, following a command sequence similar to that in Figure 17. The example system incorporates two 2764 devices, so 16K bytes of memory buffer are cleared.

Next, all the final code modules produced above (e.g., SUP130, AP130, and RJB130) must be loaded into the UPM memory buffer. The three commands in Figure 18 perform this function.

When the final system is reset, execution must branch into the root job initialization sequence. When the absolute code modules have finished loading, manually patch a jump instruction into the buffer area corresponding to the CPU reset vector. The opcode for the 8086 or 8088 intersegment jump is OEAH; the instruction's address field must contain the address assigned to label RQ$START$ADDRESS (read from the root job locate map), the 16-bit segment offset (low byte first) followed by the segment base address (ditto). The UPM CHANGE command should be used to make this patch, as illustrated in Figure 19.

The UPM memory buffer now contains a complete image of the code needed for the system EPROMs. Up until now, all software-related steps—source code preparation, translation, linking and locating—have been the same for 8086- or 8088-based systems. At this point, however, the software installation procedures diverge slightly.

Recall that the 8086 fetches instructions 16 bits at a time, from coordinated pairs of EPROMs. One contains only even-numbered program bytes, the other, odd. To separate the linear UPM buffer into high- and low-order bytes for iAPX 86/30 designs, use the UPM STRIP command as shown in Figure 20.

Now "burn" the EPROMs with the PROGRAM command in Figure 21.

```
;
;   LINK AND LOCATE THE iRMX 86 ROOT JOB
;
;   MODIFIED FOR TWO-DRIVE OPERATION
;   REVISED 10/25 - JHW
;
ASM86  :f1:RJB130 A86 MACRO(75)
;
LINK86                              &
        :f1:croot lib(rboot),       &
        :f1:RJB130 obj,             &
        :f1:croot lib               &
      TO   :f1:RJB130 lnk           &
      MAP PRINT( :f1:RJB130 mp1)
;
LOC86  :f1:RJB130 lnk               &
      TO  :f1:RJB130                &
      MAP PRINT( :f1:RJB130 mp2)    &
      OC(noli, nopl, nocm, nosb)    &
      PC(noli, pl, nocm, nosb)      &
      SEGSIZE(stack(0))             &
      ORDER(classes(data, stack, memory))      &
      ADDRESSES(classes(code(0FD150H),         &
                        data(00AD0H)))
;
OH86  :F1:RJB130 TO  :f1:RJB130 H86
;
COPY  :F1:RJB130 LST TO  LP
;
COPY  :F1:RJB130 MP1 TO  LP
;
COPY  :F1:RJB130 MP2 TO  LP
;
```

Figure 16. Root Job Configuration Commands

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
fill from 0 to 3fffh with 0ffh
```

Figure 17. UPM Initialization Sequence

```
read 86hex file : f1: sup130. h86 from 0 to 3fffh start 0fc000h
read 86hex file : f1: ap130. h86 from 0 to 3fffh start 0fc000h
read 86hex file : f1 : rjb130. h86 from 0 to 3fffh start 0fc000h
```

Figure 18. UPM Commands to Load Hex Files

```
change 3ff0h=0eah, 11h, 00h, 18h, 0fdh
```

Figure 19. UPM Command to Patch Restart Vector

```
strip low from 0 to 3fffh into 4000h
strip hi from 0 to 3fffh into 6000h
```

Figure 20. UPM Commands to Strip High and Low Bytes

```
program from 4000h to 5fffh start 0

program from 6000h to 7fffh start 0
exit
```

Figure 21. UPM Commands to Program EPROMs

To save some trouble, the UPM invocation and all commands except the manual patch can be combined into a SUBMIT command file. Replace the CHANGE command with a control-E character so the operator can adjust the starting address for the iteration. Also place control-Es before each PROGRAM step to give the operator time to socket the next memory device.

SUMMARY

The development of the 80130 marks a major milestone in the evolution of microcomputer systems. For the first time, a single VLSI device integrates the hardware facilities and operating system firmware needed by real-time multitasking applications. The 80130 offers the system hardware designer the advantages of higher integration—reduced device count, smaller boards, greater reliability—along with faster design cycles and optimal system performance.

The 80130 gives the software engineer built-in support for 35 standard operating system primitives. Application problems may now be solved at a higher level than before. It is now possible for concurrent tasks to be dispatched, memory segments allocated, and messages relayed through mailboxes nearly as easily as subroutines, dynamic variables, and I/O ports were used in the past. In effect, Jobs, Tasks, Segments, Mailboxes, and Regions become new OSP data types, manipulated entirely by firmware in the 80130.

Yet despite standardizing these functions, the OSP does not restrict the user's flexibility. The device can accommodate a variety of hardware environments, and both the hardware and software capabilities are desired.

ACKNOWLEDGEMENTS

The author would like to thank Peter Pederson for designing and implementing the demonstration system breadboard discussed in this note, Pam Johnson for her assistance in typing the manuscript, and Hal Kop, Lionel Smith, George Alexy, Chuck McMinn, and Sandy Wharton for their help in reviewing the drafts and providing many thoughtful comments and criticisms.

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX A
EXAMPLE SYSTEM SCHEMATICS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

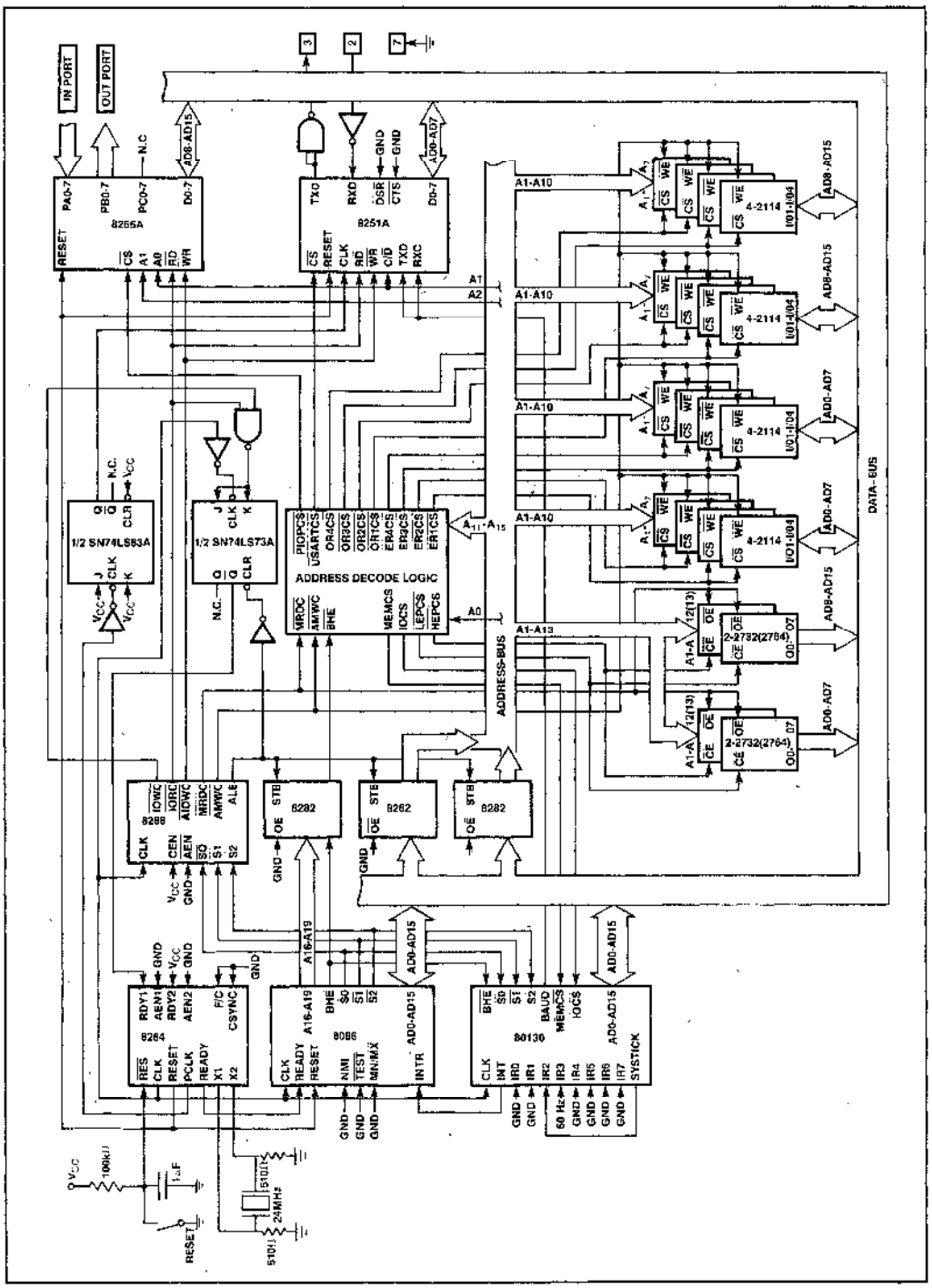


Figure A-1. Example System Schematics

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

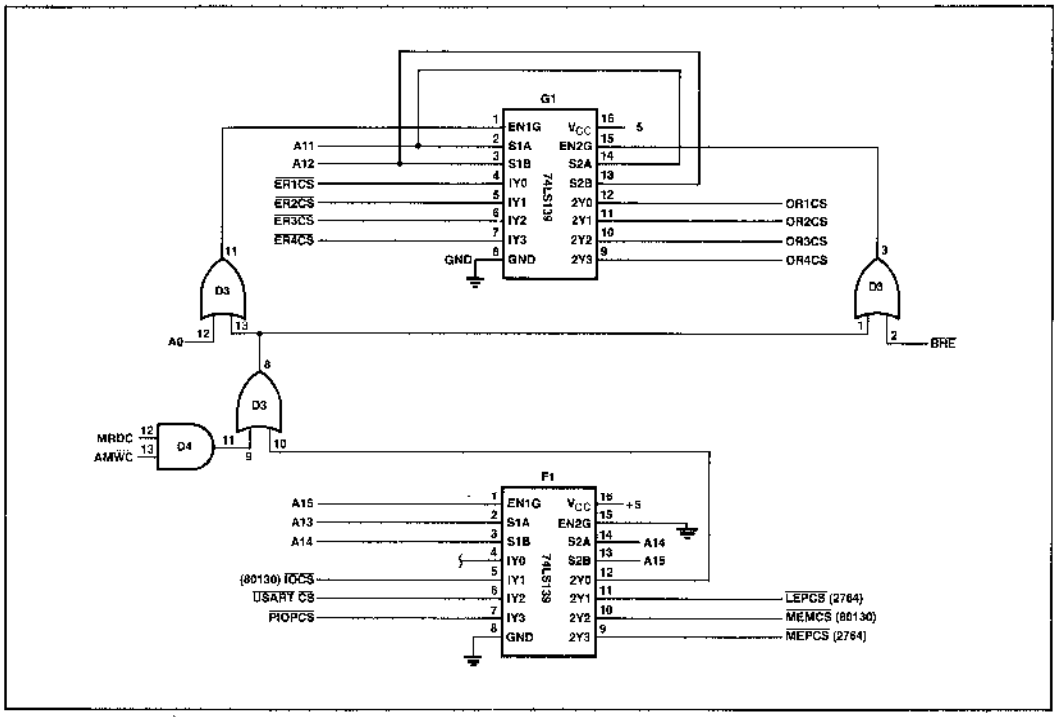


Figure A-1. Example System Schematics (continued)

AFN-02058A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX B
SOURCE CODE LISTINGS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
ISIS-II P./M-86 V2.0 COMPILATION OF MODULE DEMO130
OBJECT MODULE PLACED IN :F1:AP130.OBJ
COMPILER INVOKED BY:  PLM86  .F1:AP130.PLM  DATE(12/21)


                $DEBUG COMPACT ROM TITLE('AP-130 APPENDIX B  -  12/21/81')

   1            DEMO$130:  DO;

                /*  SYSTEM-WIDE LITERAL DECLARATIONS:  */

   2    1       DECLARE FOREVER LITERALLY 'WHILE 01H',

                /*  I/O PORT DEFINITIONS.  */

   3    1       DECLARE CHAR$51 LITERALLY '4000H',
                    CMD$51 LITERALLY '4002H',
                    STAT$51 LITERALLY '4002H',

   4    1       DECLARE PPI$A LITERALLY '6001H',
                    PPI$B LITERALLY '6003H',
                    PPI$C LITERALLY '6005H',
                    PPI$CMD LITERALLY '6007H',
                    PPI$STAT LITERALLY '6007H',

   5    1       DECLARE TIMER$CMD LITERALLY '200EH',
                    BAUD$TIMER LITERALLY '200CH';

   6    1       DECLARE AC$INTERRUPT$LEVEL LITERALLY '00111000B',

   7    1       DECLARE CR LITERALLY '0DH',
                    LF LITERALLY '0AH',
                    BEL LITERALLY '07H',

   8    1       DECLARE ASCII$CODE (16) BYTE DATA ('0123456789ABCDEF');



                $EJECT

                $INCLUDE ( :F1:NUCLUS.EXT)
            =   $SAVE NOLIST
                $INCLUDE ( :F1:NEXCEP.LIT)
            =   $save nolist

                /*  GLOBAL VARIABLE DECLARATIONS.  */

 299    1       DECLARE DATA$SEG$PTR POINTER,
                    DATA$SEG$ADDR STRUCTURE (OFFSET WORD, BASE WORD)
                        AT (@DATA$SEG$PTR),

 300    1       DECLARE HARDWARE$INIT$TASK$TOKEN WORD,
                    STATUS$TASK$TOKEN WORD,
                    MOTOR$TASK$TOKEN WORD,
                    TIME$TASK$TOKEN WORD,
                    AC$HANDLER$TOKEN WORD,
                    CRT$OUT$TASK$TOKEN WORD,
                    COMMAND$TASK$TOKEN WORD,
                    INIT$TASK$TOKEN WORD,

 301    1       DECLARE CRT$MAILBOX$TOKEN WORD,
                    CRT$REGION$TOKEN WORD;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 2  SIMPLE CRT INPUT AND OUTPUT ROUTINES.  */

302   1         C$OUT:  PROCEDURE (CHAR);
303   2             DECLARE CHAR BYTE;
304   2             DO WHILE (INPUT(STAT$51) AND 01H)=0;
                        /*  NOTHING  */
305   3                 END;
306   2             OUTPUT(CHAR$51)=CHAR;
307   2             END  C$OUT;

308   1         C$IN:  PROCEDURE BYTE;
309   2             DO WHILE (INPUT(STAT$51) AND 02H)=0;
                        /*  NOTHING  */
310   3                 END;
311   2             RETURN INPUT(CHAR$51);
312   2             END  C$IN;



                $EJECT

                /*  CODE EXAMPLE 1.  HARDWARE INITIALIZATION TASK.  */

313   1         HARDWARE$INIT$TASK.  PROCEDURE;
314   2             DECLARE HARD$INIT$EXCEPT$CODE WORD;
315   2             DECLARE PARAM$51 (*) BYTE DATA (40H, 8DH, 0QH, 40H, 4EH, 27H),
316   2             DECLARE PARAM$51$INDEX BYTE;
317   2             DECLARE SIGN$ON$MESSAGE (*) BYTE DATA
                        (CR,LF,'iAPX 86/30 HARDWARE INITIALIZED',CR,LF);
318   2             DECLARE SIGN$ON$INDEX BYTE;

319   2             OUTPUT(PPI$CMD)=90H;
320   2             OUTPUT(TIMER$CMD)=036H;
321   2             OUTPUT(BAUD$TIMER)=33;   /*GENERATES 9600 BAUD FROM 5 MHZ*/
322   2             OUTPUT(BAUD$TIMER)=0;
323   2             DO PARAM$51$INDEX=0 TO (SIZE(PARAM$51)-1);
324   3                 OUTPUT(CMD$51)=PARAM$51(PARAM$51$INDEX);
325   3                 END;     /*OF USART INITIALIZATION DO-LOOP*/
326   2             DO SIGN$ON$INDEX=0 TO (SIZE(SIGN$ON$MESSAGE)-1);
327   3                 CALL C$OUT(SIGN$ON$MESSAGE(SIGN$ON$INDEX));
328   3                 END;     /*OF SIGN-ON DO-LOOP*/
329   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@HARD$INIT$EXCEPT$CODE);
330   2             CALL RQ$DELETE$TASK(0,@HARD$INIT$EXCEPT$CODE);
331   2             END  HARDWARE$INIT$TASK;



                $EJECT

                /*  CODE EXAMPLE 3.  STATUS POLLING AND REPORTING TASK.     */


332   1         STATUS$TASK   PROCEDURE;
333   2             DECLARE STATUS$COUNTER BYTE;
334   2             DECLARE STATUS$EXCEPT$CODE WORD;

335   2             STATUS$COUNTER=0;
336   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@STATUS$EXCEPT$CODE);
337   2             DO FOREVER;
338   3                 OUTPUT(PPI$B)=INPUT(PPI$A) XOR STATUS$COUNTER;
339   3                 STATUS$COUNTER=STATUS$COUNTER+1;
340   3                 CALL RQ$SLEEP(100,@STATUS$EXCEPT$CODE);
341   3                 END;
342   2             END  STATUS$TASK;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 4.  STEPPER MOTOR CONTROL TASK.     */

343   1         DECLARE CW$STEP$DELAY BYTE,
                      CCW$STEP$DELAY BYTE,
                      CW$PAUSE$DELAY BYTE,
                      CCW$PAUSE$DELAY BYTE;

344   1         MOTOR$TASK:  PROCEDURE;
345   2             DECLARE MOTOR$EXCEPT$CODE WORD;
346   2             DECLARE MOTOR$POSITION BYTE,
                          MOTOR$PHASE BYTE;
347   2             DECLARE PHASE$CODE (4) BYTE
                          DATA (00000101B,00000110B,00001010B,00001001B);

348   2             CW$STEP$DELAY=50;     /*INITIAL STEP DELAYS = 1/4 SECOND*/
349   2             CCW$STEP$DELAY=50;
350   2             CW$PAUSE$DELAY=200,    /*PAUSES AFTER ROTATION = 1 SECOND*/
351   2             CCW$PAUSE$DELAY=200;
352   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@MOTOR$EXCEPT$CODE);
353   2             DO FOREVER;
354   3                 DO MOTOR$POSITION=0 TO 100;
355   4                     MOTOR$PHASE=MOTOR$POSITION AND 0003H;
356   4                     OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE);
357   4                     CALL RQ$SLEEP(CW$STEP$DELAY,@MOTOR$EXCEPT$CODE);
358   4                     END;
359   3                 CALL RQ$SLEEP(CW$PAUSE$DELAY,@MOTOR$EXCEPT$CODE);
360   3                 DO MOTOR$POSITION=0 TO 100;
361   4                     MOTOR$PHASE=(100-MOTOR$POSITION) AND 0003H;
362   4                     OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE);
363   4                     CALL RQ$SLEEP(CCW$STEP$DELAY,@MOTOR$EXCEPT$CODE);
364   4                     END;
365   3                 CALL RQ$SLEEP(CCW$PAUSE$DELAY,@MOTOR$EXCEPT$CODE);
366   3                 END;
367   2             END  MOTOR$TASK;



                $EJECT

                /*  CODE EXAMPLE 5.  INTERRUPT HANDLER TO TRACK 60 HZ INPUT.  */

368   1         DECLARE AC$CYCLE$COUNT BYTE;

369   1         AC$HANDLER:  PROCEDURE INTERRUPT 59;     /*VECTOR FOR 80130 INT3*/
370   2             DECLARE AC$EXCEPT$CODE WORD;

371   2             CALL RQ$ENTER$INTERRUPT(AC$INTERRUPT$LEVEL,@AC$EXCEPT$CODE);
372   2             AC$CYCLE$COUNT=AC$CYCLE$COUNT+1;
373   2             IF AC$CYCLE$COUNT >= 60
                      THEN DO;
375   3                     AC$CYCLE$COUNT=0;
376   3                     CALL RQ$SIGNAL$INTERRUPT(AC$INTERRUPT$LEVEL,
                                @AC$EXCEPT$CODE);
377   3                     END;
378   2                 ELSE CALL RQ$EXIT$INTERRUPT(AC$INTERRUPT$LEVEL,
                              @AC$EXCEPT$CODE);
379   2             END  AC$HANDLER;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 7.  PROTECTED CRT OUTPUT SUBROUTINE.  */

380   1         PROTECTED$CRT$OUT:  PROCEDURE (CHAR) REENTRANT;
381   2             DECLARE CHAR BYTE;
382   2             DECLARE CRT$EXCEPT$CODE WORD;
383   2             CALL RQ$RECEIVE$CONTROL(CRT$REGION$TOKEN,@CRT$EXCEPT$CODE);
384   2             DO WHILE (INPUT(STAT$51) AND 01H)=0;
                        /*  NOTHING  */
385   3                   END;
386   2             OUTPUT(CHAR$51)=CHAR;
387   2             CALL RQ$SEND$CONTROL(@CRT$EXCEPT$CODE);
388   2         END  PROTECTED$CRT$OUT;


                $EJECT

                /*  CODE EXAMPLE 6.  INTERRUPT TASK TO MONITOR CLOCK TIME.  */

389   1         DECLARE SECOND$COUNT BYTE,
                      MINUTE$COUNT BYTE,
                      HOUR$COUNT BYTE;

390   1         TIME$TASK:  PROCEDURE;
391   2             DECLARE TIME$EXCEPT$CODE WORD;

392   2             AC$CYCLE$COUNT=0;
393   2             CALL RQ$SET$INTERRUPT(AC$INTERRUPT$LEVEL,01H,
                          INTERRUPT$PTR(AC$HANDLER),DATA$SEG$ADDR,BASE,
                          @TIME$EXCEPT$CODE);
394   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@TIME$EXCEPT$CODE);
395   2             DO HOUR$COUNT=0 TO 23;
396   3                 DO MINUTE$COUNT=0 TO 59;
397   4                     DO SECOND$COUNT=0 TO 59;
398   5                         CALL RQ$WAIT$INTERRUPT(AC$INTERRUPT$LEVEL,
                                      @TIME$EXCEPT$CODE);
399   5                         IF SECOND$COUNT MOD 5 = 0
                                  THEN CALL PROTECTED$CRT$OUT(BEL);
401   5                     END;     /*  SECOND LOOP  */
402   4                 END;     /*  MINUTE LOOP  */
403   3             END;     /*  HOUR LOOP  */
404   2             CALL RQ$RESET$INTERRUPT(AC$INTERRUPT$LEVEL,
                          @TIME$EXCEPT$CODE);
405   2             CALL RQ$DELETE$TASK(0,@TIME$EXCEPT$CODE);
406   2             END  TIME$TASK;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 8.  SUBROUTINE TO CREATE TIME-OF-DAY MESSAGE.  */

407     1       PRINT$TOD:  PROCEDURE;
408     2           DECLARE TOD$MESSAGE$TOKEN WORD;
409     2           DECLARE TOD$EXCEPT$CODE WORD;
410     2           DECLARE TOD$SEGMENT$OFFSET WORD,
                        TOD$SEGMENT$BASE WORD;
411     2           DECLARE TOD$SEGMENT$PNTR POINTER AT (@TOD$SEGMENT$OFFSET);
412     2           DECLARE TOD$TEMPLATE (28) BYTE
                        DATA (27,'THE TIME IS NOW hh:mm:ss.  ',CR,LF);
413     2           DECLARE TOD$STRING BASED TOD$SEGMENT$PNTR (28) BYTE;
414     2           DECLARE TOD$STRING$INDEX BYTE;

415     2           TOD$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(28,@TOD$EXCEPT$CODE);
416     2           TOD$SEGMENT$BASE=TOD$MESSAGE$TOKEN;
417     2           TOD$SEGMENT$OFFSET=0;
418     2           DO TOD$STRING$INDEX=0 TO 27;
419     3               TOD$STRING(TOD$STRING$INDEX)=
                            TOD$TEMPLATE(TOD$STRING$INDEX);
420     3               END;
421     2           TOD$STRING(17)=ASCII$CODE(HOUR$COUNT/10);
422     2           TOD$STRING(18)=ASCII$CODE(HOUR$COUNT MOD 10);
423     2           TOD$STRING(20)=ASCII$CODE(MINUTE$COUNT/10);
424     2           TOD$STRING(21)=ASCII$CODE(MINUTE$COUNT MOD 10);
425     2           TOD$STRING(23)=ASCII$CODE(SECOND$COUNT/10);
426     2           TOD$STRING(24)=ASCII$CODE(SECOND$COUNT MOD 10);
427     2           CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
                        TOD$MESSAGE$TOKEN,0,@TOD$EXCEPT$CODE);
428     2           RETURN;
429     2           END  PRINT$TOD;


                $EJECT

                /*  CODE EXAMPLE 9.  SUBROUTINE TO CREATE SWITCH STATUS MESSAGE.  */

430     1       PRINT$STATUS:  PROCEDURE;
431     2           DECLARE STATUS$MESSAGE$TOKEN WORD;
432     2           DECLARE STATUS$EXCEPT$CODE WORD;
433     2           DECLARE STATUS$SEGMENT$OFFSET WORD,
                        STATUS$SEGMENT$BASE WORD;
434     2           DECLARE STATUS$SEGMENT$PNTR POINTER
                        AT (@STATUS$SEGMENT$OFFSET);
435     2           DECLARE STATUS$TEMPLATE (40) BYTE DATA
                        (39,'THE SWITCHES ARE NOW SET TO .... ....B',CR,LF);
436     2           DECLARE STATUS$STRING BASED STATUS$SEGMENT$PNTR (40) BYTE;
437     2           DECLARE STATUS$STRING$INDEX BYTE;
438     2           DECLARE BIT$PATTERN BYTE;

439     2           STATUS$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(40,
                        @STATUS$EXCEPT$CODE);
440     2           STATUS$SEGMENT$BASE=STATUS$MESSAGE$TOKEN;
441     2           STATUS$SEGMENT$OFFSET=0;
442     2           DO STATUS$STRING$INDEX=0 TO 39;
443     3               STATUS$STRING(STATUS$STRING$INDEX)=
                            STATUS$TEMPLATE(STATUS$STRING$INDEX);
444     3               END;
445     2           BIT$PATTERN=INPUT(PPI$A);
446     2           DO STATUS$STRING$INDEX=29 TO 36;
447     3               STATUS$STRING(STATUS$STRING$INDEX)=
                            ASCII$CODE(BIT$PATTERN AND 01H);
448     3               BIT$PATTERN=ROR(BIT$PATTERN,1);
449     3               END;
450     2           CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
                        STATUS$MESSAGE$TOKEN,0,@STATUS$EXCEPT$CODE);
451     2           END  PRINT$STATUS;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 10.  TASK TO RECEIVE MESSAGES AND TRANSMIT THEM TO CRT.  */

452   1         CRT$OUT$TASK:  PROCEDURE;
453   2             DECLARE MESSAGE$LENGTH BYTE;
454   2             DECLARE MESSAGE$TOKEN WORD;
455   2             DECLARE RESPONSE$TOKEN WORD;
456   2             DECLARE MESSAGE$EXCEPT$CODE WORD;
457   2             DECLARE MESSAGE$SEGMENT$OFFSET WORD,
                          MESSAGE$SEGMENT$BASE WORD;
458   2             DECLARE MESSAGE$SEGMENT$PNTR POINTER AT (@MESSAGE$SEGMENT$OFFSET);
459   2             DECLARE MESSAGE$STRING$CHAR BASED MESSAGE$SEGMENT$PNTR BYTE;

460   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@MESSAGE$EXCEPT$CODE);
461   2             DO FOREVER;
462   3                 MESSAGE$TOKEN=RQ$RECEIVE$MESSAGE(CRT$MAILBOX$TOKEN,0FFFFH,
                              @RESPONSE$TOKEN,@MESSAGE$EXCEPT$CODE);
463   3                 MESSAGE$SEGMENT$OFFSET=0;
464   3                 MESSAGE$SEGMENT$BASE=MESSAGE$TOKEN;
465   3                 MESSAGE$LENGTH=MESSAGE$STRING$CHAR;
466   3                 DO MESSAGE$SEGMENT$OFFSET=1 TO MESSAGE$LENGTH;
467   4                     CALL PROTECTED$CRT$OUT(MESSAGE$STRING$CHAR);
468   4                     END;
469   3                 CALL RQ$DELETE$SEGMENT(MESSAGE$TOKEN,@MESSAGE$EXCEPT$CODE);
470   3                 END;    /*  OF FOREVER-LOOP  */
471   2             END  CRT$OUT$TASK;


                $EJECT

                /*  CODE EXAMPLE 11.  TASK TO POLL KEYBOARD AND PROCESS COMMANDS.  */

472   1         COMMAND$TASK:  PROCEDURE;
473   2             DECLARE CONSOLE$CHAR BYTE;
474   2             DECLARE COMMAND$EXCEPT$CODE WORD;

475   2             CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@COMMAND$EXCEPT$CODE);
476   2             DO FOREVER;
477   3                 CONSOLE$CHAR=C$IN AND 7FH;
478   3                 CALL PROTECTED$CRT$OUT(CONSOLE$CHAR);
479   3                 IF CONSOLE$CHAR=CR
                              THEN CALL PROTECTED$CRT$OUT(LF);
481   3                 IF (CONSOLE$CHAR >= '0') AND (CONSOLE$CHAR <= '9')
                              THEN DO;
483   4                         CALL PROTECTED$CRT$OUT(CR);
484   4                         CALL PROTECTED$CRT$OUT(LF);
485   4                         DO CASE (CONSOLE$CHAR-'0');
486   5                             CALL PRINT$TOD;
487   5                             CALL PRINT$STATUS;
488   5                             CALL RQ$SUSPEND$TASK(CRT$OUT$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
489   5                             CALL RQ$RESUME$TASK(CRT$OUT$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
490   5                             CALL RQ$DISABLE(AC$INTERRUPT$LEVEL,
                                          @COMMAND$EXCEPT$CODE);
491   5                             CALL RQ$ENABLE(AC$INTERRUPT$LEVEL,
                                          @COMMAND$EXCEPT$CODE);
492   5                             CALL RQ$SUSPEND$TASK(MOTOR$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
493   5                             CALL RQ$RESUME$TASK(MOTOR$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
494   5                             CALL RQ$SUSPEND$TASK(STATUS$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
495   5                             CALL RQ$RESUME$TASK(STATUS$TASK$TOKEN,
                                          @COMMAND$EXCEPT$CODE);
496   5                             END;    /*  OF CASE-LIST  */
497   4                         END;    /*  OF COMMAND PROCESSING  */
498   3                 END;
499   2             END  COMMAND$TASK;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                $EJECT

                /*  CODE EXAMPLE 12.  TASK TO INITIALIZE DSP SOFTWARE.  */

500   1         INIT$TASK: PROCEDURE PUBLIC;
501   2             DECLARE INIT$EXCEPT$CODE WORD;

502   2             DATA$SEG$PTR=@INIT$TASK$TOKEN;   /*LOAD DATA SEGMENT BASE*/
503   2             CRT$MAILBOX$TOKEN=RQ$CREATE$MAILBOX(0,@INIT$EXCEPT$CODE);
504   2             CRT$REGION$TOKEN=RQ$CREATE$REGION(0,@INIT$EXCEPT$CODE);
505   2             INIT$TASK$TOKEN=RQ$GET$TASK$TOKENS(0,@INIT$EXCEPT$CODE);
506   2             HARDWARE$INIT$TASK$TOKEN=RQ$CREATE$TASK
                          (110,@HARDWARE$INIT$TASK,DATA$SEG$ADDR.BASE,0,300,
                          0,@INIT$EXCEPT$CODE);
507   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
508   2             STATUS$TASK$TOKEN=RQ$CREATE$TASK(110,@STATUS$TASK,
                          DATA$SEG$ADDR.BASE,0,300,0,@INIT$EXCEPT$CODE);
509   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
510   2             MOTOR$TASK$TOKEN=RQ$CREATE$TASK(110,@MOTOR$TASK,
                          DATA$SEG$ADDR.BASE,0,300,0,@INIT$EXCEPT$CODE);
511   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
512   2             TIME$TASK$TOKEN=RQ$CREATE$TASK(120,@TIME$TASK,
                          DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE);
513   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
514   2             CRT$OUT$TASK$TOKEN=RQ$CREATE$TASK(120,@CRT$OUT$TASK,
                          DATA$SEG$ADDR.BASE,0,300,0,@INIT$EXCEPT$CODE);
515   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
516   2             COMMAND$TASK$TOKEN=RQ$CREATE$TASK(130,@COMMAND$TASK,
                          DATA$SEG$ADDR BASE,0,300,0,@INIT$EXCEPT$CODE);
517   2             CALL RQ$SUSPEND$TASK(0,@INIT$EXCEPT$CODE);
518   2             CALL RQ$END$INIT$TASK;
519   2             CALL RQ$DELETE$TASK(0,@INIT$EXCEPT$CODE);
520   2             END  INIT$TASK;


521   1         END  DEMO$130;


MODULE INFORMATION

     CODE AREA SIZE     = 084CH   2124D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0052H     82D
     MAXIMUM STACK SIZE = 0026H     38D
     848 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX C
SYSTEM MEMORY MAP

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXAMPLE SYSTEM MEMORY MAP

	MEMORY MODULE	STARTING ADDRESS	ENDING ADDRESS
EPROM (2x2764)	8086 RESTART VECTOR	0FFFF:0	0FFFF:F
	ROOT JOB CODE AREA	0FD18:0	0FD36:5
	APPLICATION JOB CODE AREA	0FC62:0	0FD17:B
	OSP SUPPORT CODE AREA	0FC00:0	0FC61:F
	80130 MEMORY SPACE	0F800:0	0FBFF:F
RAM	(FREE SYSTEM RAM)	00C0:0	01FF:F
	ROOT JOB DATA AREA	00AD:0	00BF:F
	APPLICATION JOB DATA AREA	00A7:0	00AC:1
	OSP SUPPORT DATA AREA	0040:0	00A6:F
	8088 INTERRUPT VECTOR	0000:0	003F:F

INITIALIZATION TASK STARTING ADDRESS: FC62:06B5

ROOT JOB STARTING ADDRESS: FD18:0011

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX D
SUPPORT CODE LOCATE MAP

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
ISIS-II MCS-86 LOCATER, V1.2 INVOKED BY:
 F0 LOC86                                                &
 F1 SUP130 LNK TO  F1 SUP130 MAP PRINT( F1 SUP130 MP2) SG(3) &
SEGSIZE(STACK(0))                                        &
ADDRESSES(CLASSES(CODE(0F800H),DATA(00400H)))            &
ORDER(CLASSES(DATA,STACK))                               &
OBJECTCONTROLS(NOLINES,NOCOMMENTS,NOSYMBOLS)
WARNING 26   DECREASING SIZE OF SEGMENT
  SEGMENT   STACK

SYMBOL TABLE OF MODULE MINIMAL_B0130
READ FROM FILE  F1 SUP130 LNK
WRITTEN TO FILE  F1 SUP130

BASE   OFFSET TYPE SYMBOL                BASE   OFFSET TYPE SYMBOL                BASE   OFFSET TYPE SYMBOL

0040H  0000H  PUB  INTERRUPTTASKVEC      0040H  0120H  PUB  DEFAULT_HANDLER      0040H  0144H  PUB  READYLISTROOT
0040H  0148H  PUB  INTERRORENTRY         0040H  014CH  PUB  SYSTEMEXCEPTIONH     0040H  0150H  PUB  DELETIONTASKTOKE
                                                           -ANDLERPTR                                -N
0040H  0152H  PUB  EXTENSIONLISTROO      0040H  0154H  PUB  DELETION_OBJECT_     0040H  0156H  PUB  SYSTEMPOOLTOKEN
                  -T                                       -BASE
0040H  0158H  PUB  ROOTJOBTOKEN          0040H  015AH  PUB  MINTRANSSIZE         0040H  015CH  PUB  LAST_NDP_TASK
0040H  015EH  PUB  NDP_INTERRUPT_LE      0040H  0160H  PUB  PARAM_VALIDATION     0040H  0162H  PUB  REGION_FLAGS
                  -VEL_VAR                                 -_VECTOR
0040H  0164H  PUB  TASK_WAITING_FLA      0040H  0166H  PUB  REGION_TOKEN_TAB     0040H  0176H  PUB  SIGNAL_0_INDEX
                  -GS                                      -LE
0040H  0178H  PUB  SIGNAL_0              0040H  01E8H  PUB  KERNEL_FLAG          0040H  01E9H  PUB  ACTIVATE_SIGNAL_
                                                                                                   -0
0040H  01EAH  PUB  FILLCHAR              0040H  01EBH  PUB  NUM_SLAVES           0040H  01ECH  PUB  OLD_SLAVE_NUM
0040H  01EDH  PUB  INTMASK               0040H  01F6H  PUB  DISABLEMASK          0040H  01FFH  PUB  LEVEL_SET_TABLE
0040H  020BH  PUB  IMR_PORT              0040H  021AH  PUB  EOI_PORT             0040H  022CH  PUB  ISR_PORT
0040H  023EH  PUB  PIC_INFO              0040H  0247H  PUB  CLOCK_SPEC_EOI       0040H  024BH  PUB  CLOCK_ON
0040H  0249H  PUB  CLOCK_OFF             0040H  024AH  PUB  CLOCK_LEVEL          0040H  05D0H  PUB  END_OF_DATA
F800H  45CCH  PUB  NDP_INTERRUPT_LE      F800H  45C2H  PUB  VALIDATE_PARAMS_     F800H  4542H  PUB  GETDESCRTOKEN
                  -VEL                                     -BODY_DUMMY
F800H  4556H  PUB  GETDESCRPOINTER       F800H  4547H  PUB  GETPOINTER          F800H  453DH  PUB  SCANMEMORY
F800H  4538H  PUB  OVERFLOW              F800H  4533H  PUB  NENTRY_BODY         F800H  452EH  PUB  KSUSPEND
F800H  4529H  PUB  KINITIALIZE           F800H  4524H  PUB  KENABLELEVELNS      F800H  451FH  PUB  KENABLELEVEL
F800H  451AH  PUB  KCREATEREGIONNS       F800H  4515H  PUB  KCREATEOBJECTNS     F800H  4510H  PUB  KCREATEOBJECT
F800H  4508H  PUB  INITNDP               F800H  4506H  PUB  INITIALIZE_..       F800H  4501H  PUB  FINISHINITIALIZA
                                                                                                   -TION
F800H  44FCH  PUB  EOI_ROUTINE           F800H  44F7H  PUB  DIVIDEBYZERO          F800H  44F2H  PUB  DECODE_LEVEL
F800H  44EDH  PUB  COMMON_ERROR          F800H  44E8H  PUB  CLOCKENTRY_BODY       F800H  44E3H  PUB  ARRAYBOUNDS
F800H  44D0H  PUB  SYSTEMEXCEPTIONH      F800H  4472H  PUB  INITIALIZE_TIMER      F800H  43AEH  PUB  INITIALIZE_PICS
                  -ANDLER
F800H  435CH  PUB  INIT_INTERNAL_RE      F800H  434EH  PUB  NDP_INTERRUPT_HA     F800H  433FH  PUB  CLOCKENTRY
                  -GIONS                                   -NDLER
F800H  4336H  PUB  NENTRY                F800H  40FEH  PUB  INITIALIZENUCLEU     F800H  4086H  PUB  RQWAITINTERRUPT_
                                                           -S                                      -BODY
F800H  40B1H  PUB  RQSIGNALINTERRUP      F800H  40ACH  PUB  RQGETLEVEL_BODY      F800H  40A7H  PUB  RQEXITINTERRUPT_
                  -T_BODY                                                                          -BODY
F800H  40A2H  PUB  RQENTERINTERRUPT      F800H  409DH  PUB  RQDISABLE_BODY      F800H  4094H  PUB  RQWAITINTERRUPT
                  -_BODY
F800H  408AH  PUB  RQSIGNALINTERRUP      F800H  40B0H  PUB  RQGETLEVEL          F800H  4076H  PUB  RQENTERINTERRUPT
                  -T
F800H  406CH  PUB  RQEXITINTERRUPT       F800H  4062H  PUB  RQDISABLE           F800H  405DH  PUB  NUNLOCK_DELETION
                                                                                                   -_OBJECT
F800H  4058H  PUB  NUNLOCKNS             F800H  4053H  PUB  NUNLOCK             F800H  404EH  PUB  NOPEN_DELETION_O
                                                                                                   -BJECT
F800H  4049H  PUB  NOPENNS               F800H  4044H  PUB  NOPEN              F800H  403FH  PUB  NLOCK_DELETION_O
                                                                                                   -BJECT
F800H  403AH  PUB  NLOCKNS               F800H  4035H  PUB  NLOCK              F800H  4030H  PUB  NCLOSE_DELETION_
                                                                                                   -OBJECT
F800H  402BH  PUB  NCLOSENS              F800H  4026H  PUB  NCLOSE             F800H  4021H  PUB  DELETERUNNINGTAS
                                                                                                   -K
F800H  401CH  PUB  DELETEOBJECT          F800H  400AH  PUB  COPYRIGHT          F800H  4000H  PUB  NBEGIN
F800H  4000H  PUB  INIT_NUCLEUS_JUM      FC5DH  0004H  PUB  IMR_START         FC5CH  000EH  PUB
                  -P
FC5CH  000FH  PUB  INIT_CMD1             FC5CH  0010H  PUB  INIT_CMDS_MASTER   FC5CH  0011H  PUB
FC5CH  0012H  PUB  INIT_CMD4_MASTER      FC61H  000EH  PUB  SLAVE_TABLE        FC61H  0003H  PUB

FC61H  0005H  PUB  CLOCK_0_PORT          FC61H  0007H  PUB  CLOCK_COUNT        FC61H  000AH  PUB
FC61H  000BH  PUB  C_CLOCK_SPEC_EOI      FC61H  000CH  PUB  C_CLOCK_ON         FC61H  0009H  PUB
F800H  4576H  PUB  LEVEL7_HANDLER        F800H  4574H  PUB  PARAM_VALIDATION
                                                           -_PATH

MEMORY MAP OF MODULE MINIMAL_B0130
READ FROM FILE  F1 SUP130 LNK
WRITTEN TO FILE  F1 SUP130

SEGMENT MAP

START     STOP      LENGTH ALIGN NAME            CLASS

00000H    003FFH     0400H  A    (ABSOLUTE)
00400H    009EFH     05F0H  W    DATA            DATA
009F0H    009FFH     0010H  G    INTVEC_REG_SEG  DATA
00A00H    00A0FH     0010H  G    EXT_REG_SEG     DATA
00A10H    00A1FH     0010H  G    JOB_REG_SEG     DATA
00A20H    00A2FH     0010H  G    SEM_REG_SEG     DATA
00A30H    00A3FH     0010H  G    MAIL_REG_SEG    DATA
00A40H    00A4FH     0010H  G    OD_REG_SEG      DATA
00A50H    00A5FH     0010H  G    POOL_REG_SEG    DATA
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
00A60H   00A6FH    0010H   G   DELETION_REG_S    DATA      <-- LAST RAM BYTE USED
                               -EG

00A70H   00A70H    0000H   W   STACK            STACK
00A70H   00A70H    0000H   G   ??SEG
FC00H    FC5DH     45DEH   W   CODE             CODE
FC5EH    FC5D2H    0009H   W   PIC_CNF_SEG      CODE
FC5D4H   FC5E5H    0012H   W   _IMR_PORT        CODE
FC5E6H   FC5F7H    0012H   W   _EOI_PORT        CODE
FC5F8H   FC609H    0012H   W   _ISR_READ_PORT   CODE
FC60AH   FC612H    0009H   B   _PIC_INFO        CODE
FC613H   FC61CH    000AH   B   TIMER_CNF_SEG    CODE
FC61EH   FC61FH    0000H   W   CSEG             CODE

FC61EH   FC61FH    0002H   W   SLAVE_SEG        CODE      <-- LAST EPROM BYTE USED

FC620H   FC620H    0000H   W   MEMORY           MEMORY


GROUP MAP

ADDRESS  GROUP OR SEGMENT NAME
0040DH   DGROUP
         DATA
         INTVEC_REQ_SEG
         EXT_REQ_SEG
         JOB_REQ_SEG
         SEM_REQ_SEG
         MAIL_REQ_SEG
         DD_REQ_SEG
         POOL_REQ_SEG
         DELETION_REG_SEG
FC00H    CGROUP
         CODE
         PIC_CNF_SEG
         _IMR_PORT
         _EOI_PORT
         _ISR_READ_PORT
         _PIC_INFO
         TIMER_CNF_SEG
         CSEG


         SLAVE_SEG
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX E
APPLICATION JOB LOCATE MAP

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
ISIS-II MCS-86 LOCATER, V1.2 INVOKED BY
LOC86  F1 AP130 LNK TO  F1 AP130                          &
    ORDER (CLASSES(DATA,STACK,MEMORY))                    &
    SEGSIZE (STACK (0))                                   &
    ADDRESSES (CLASSES (DATA (00A70H),                    &
                        CODE (0FC620H)))                  &
    MAP PRINT ( F1 AP130 MP2)                             &
    OBJECTCONTROLS (NOLINES,NOCOMMENTS,NOPUBLICS,NOSYMBOLS)
WARNING 26   DECREASING SIZE OF SEGMENT
  SEGMENT   STACK

SYMBOL TABLE OF MODULE DEMO130
READ FROM FILE  F1 AP130 LNK
WRITTEN TO FILE  F1 AP130

BASE   OFFSET TYPE SYMBOL                BASE   OFFSET TYPE SYMBOL

FC62H  0B3AH  PUB  RQENDINITTASK         FC62H  0B1CH  PUB  RQ_N_C_RETURN_40
FC62H  0B00H  PUB  RQ_N_C_RETURN_20      FC62H  0AE4H  PUB  RQ_N_C_RETURN_14
FC62H  0AC8H  PUB  RQ_N_C_RETURN_12      FC62H  0AACH  PUB  RQ_N_C_RETURN_10
FC62H  0A90H  PUB  RQ_N_C_RETURN_8       FC62H  0A74H  PUB  RQ_N_C_RETURN_6
FC62H  0A58H  PUB  RQ_N_C_RETURN_4       FC62H  0A3EH  PUB  RQERROR
FC62H  0A28H  PUB  RQGETLEVEL            FC62H  0A0EH  PUB  RQSIGNALEXCEPTIO
                                                          -N
FC62H  09F0H  PUB  RQWAITINTERRUPT       FC62H  09DAH  PUB  RQSIGNALINTERRUP
                                                          -T
FC62H  09D4H  PUB  RQDELETESEMAPHOR      FC62H  09CEH  PUB  RQDELETEMAILBOX
                   -E
FC62H  09B8H  PUB  RQEXITINTERRUPT       FC62H  0932H  PUB  RQUNCATALOGOBJEC
                                                          -T
FC62H  09ACH  PUB  RQSENDUNITS           FC62H  09A6H  PUB  RQSUSPENDTASK
FC62H  09A0H  PUB  RQSETPRIORITY         FC62H  099AH  PUB  RQSETPOOLMIN
FC62H  0974H  PUB  RQGETOSEXTENSION      FC62H  095EH  PUB  RQSENDMESSAGE
FC62H  0988H  PUB  RQSLEEP               FC62H  0982H  PUB  RQSETINTERRUPT
FC62H  097CH  PUB  RQSETEXCEPTIONHA      FC62H  0976H  PUB  RQGETOSCONTROL
                   -NDLER
FC62H  0970H  PUB  RQRECEIVEUNITS        FC62H  096AH  PUB  RQRESUMETASK
FC62H  0938H  PUB  RQRECEIVEMESSAGE      FC62H  0932H  PUB  RQRESETINTERRUPT
FC62H  092CH  PUB  RQRECEIVECONTROL      FC62H  0926H  PUB  RQOFFSPRING
FC62H  0920H  PUB  RQLOOKUPOBJECT        FC62H  091AH  PUB  RQINSPECTCOMPOSI
                                                          TE
FC62H  0914H  PUB  RQGETTASKTOKENS       FC62H  090EH  PUB  RQGETTYPE
FC62H  0908H  PUB  RQGETSIZE             FC62H  0902H  PUB  RQGETPRIORITY
FC62H  08FCH  PUB  RQGETPOOLATTRIB       FC62H  08F6H  PUB  RQGETEXCEPTIONHA
                                                          -NDLER
FC62H  08F0H  PUB  RQFORCEDELETE         FC62H  08EAH  PUB  RQENABLE
FC62H  08D4H  PUB  RQENTERINTERRUPT      FC62H  08CEH  PUB  RQENABLEDELETION
FC62H  08C8H  PUB  RQDELETETASK          FC62H  08C2H  PUB  RQDELETESEGMENT
FC62H  08ACH  PUB  RQDISABLE             FC62H  08A6H  PUB  RQDELETEREGION
FC62H  08A0H  PUB  RQDELETEJOB           FC62H  089AH  PUB  RQDELETEEXTENSIO
                                                          -N
FC62H  0894H  PUB  RQDISABLEDELETIO      FC62H  088EH  PUB  RQDELETECOMPOSIT
                   -N                                     -E
FC62H  0888H  PUB  RQCREATETASK          FC62H  0882H  PUB  RQCREATESEMAPHOR
                                                          -E
FC62H  087CH  PUB  RQCREATESEGMENT       FC62H  0876H  PUB  RQCREATEREGION
FC62H  0870H  PUB  RQCATALOGOBJECT       FC62H  086AH  PUB  RQCREATEMAILBOX
FC62H  0864H  PUB  RQCREATEJOB           FC62H  085EH  PUB  RQCREATEEXTENSIO
                                                          -N
FC62H  0858H  PUB  RQCREATECOMPOSIT      FC62H  0852H  PUB  RQALTERCOMPOSITE
                   -E
FC62H  084CH  PUB  RQACCEPTCONTROL       FC62H  0685H  PUB  INITTASK

DEMO130   SYMBOLS AND LINES
FD17H  000CH  SYM  MEMORY                FC62H  0000H  SYM  ASCIICODE
00A7H  0000H  SYM  DATASEGPTR            00A7H  0000H  SYM  DATASEGADDR
00A7H  0004H  SYM  HARDWAREINITTASK      00A7H  0006H  SYM  STATUSTASKTOKEN
                   -TOKEN
00A7H  0008H  SYM  MOTORTASKTOKEN        00A7H  000AH  SYM  TIMETASKTOKEN
00A7H  000CH  SYM  ACHANDLERTOKEN        00A7H  000EH  SYM  CRTOUTTASKTOKEN
00A7H  0010H  SYM  COMMANDTASKTOKEN      00A7H  0012H  SYM  INITTASKTOKEN
00A7H  0014H  SYM  CRTMAILBOXTOKEN       00A7H  0016H  SYM  CRTREGIONTOKEN
FC62H  0084H  SYM  COUT                  STACK  0004H  SYM  CHAR
FC62H  00A1H  SYM  CIN                   FC62H  00B9H  SYM  HARDWAREINITTASK
00A7H  001BH  SYM  HARDINITEXCEPTCO      FC62H  0010H  SYM  PARAM51
                   -DE
00A7H  0040H  SYM  PARAM51INDEX          FC62H  0016H  SYM  SIGNONMESSAGE
00A7H  0041H  SYM  SIGNONINDEX           FC62H  013BH  SYM  STATUSTASK
00A7H  0042H  SYM  STATUSCOUNTER         00A7H  001AH  SYM  STATUSEXCEPTCODE
00A7H  0043H  SYM  CWSTEPDELAY           00A7H  0044H  SYM  CCWSTEPDELAY
00A7H  0045H  SYM  CWPAUSEDELAY          00A7H  0046H  SYM  CCWPAUSEDELAY
FC62H  0172H  SYM  MOTORTABK             00A7H  001CH  SYM  MOTOREXCEPTCODE
00A7H  0047H  SYM  MOTORPOSITION         00A7H  0048H  SYM  MOTORPHASE
FC62H  0039H  SYM  PHASECODE             00A7H  0049H  SYM  ACCYCLECOUNT
FC62H  0256H  SYM  ACHANDLER             00A7H  001EH  SYM  ACEXCEPTCODE
FC62H  029CH  SYM  PROTECTEDCRTOUT       STACK  0006H  SYM  CHAR
STACK  0002H  SYM  CRTEXCEPTCODE         00A7H  004AH  SYM  SECONDCOUNT
00A7H  004BH  SYM  MINUTECOUNT           00A7H  004CH  SYM  HOURCOUNT
FC62H  02CFH  SYM  TIMETASK              00A7H  0020H  SYM  TIMEEXCEPTCODE
FC62H  038BH  SYM  PRINTTOD              00A7H  0022H  SYM  TODMESSAGETOKEN
00A7H  0024H  SYM  TODEXCEPTCODE         00A7H  0026H  SYM  TODSEGMENTOFFSET
00A7H  0028H  SYM  TODSEGMENTBASE        00A7H  002AH  SYM  TODSEGMENTPNTR
FC62H  003DH  SYM  TODTEMPLATE           00A7H  0026H  BAS  TODSTRING
00A7H  004DH  SYM  TODSTRINGINDEX        FC62H  0489H  SYM  PRINTSTATUS
00A7H  002AH  SYM  STATUSMESSAGETOK      00A7H  002CH  SYM  STATUSEXCEPTCODE
                   -EN
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
00A7H  002EH  SYM  STATUSSEGMENTOFF     00A7H  0036H  SYM  STATUSSEGMENTBAS
                   -SET                                    -E
00A7H  002EH  SYM  STATUSSEGMENTPNT     FC62H  0059H  SYM  STATUSTEMPLATE
                   -R
00A7H  002EH  BAS  STATUSSTRING         00A7H  004EH  SYM  STATUSSTRINGINDE
                                                           -X
00A7H  004FH  SYM  BITPATTERN           FC62H  052FH  SYM  CRTOUTTASK
00A7H  0050H  SYM  MESSAGELENGTH        00A7H  0032H  SYM  MESSAGETOKEN
00A7H  0034H  SYM  RESPONSETOKEN        00A7H  0036H  SYM  MESSAGEEXCEPTCOD
                                                           -E
00A7H  003BH  SYM  MESSAGESEGMENTOF     00A7H  003AH  SYM  MESSAGESEGMENTBA
                   -FSET                                   -SE
00A7H  0050H  SYM  MESSAGESEGMENTPN     00A7H  0030H  BAS  MESSAGESTRINGIN
                   -TR                                     -R
FC62H  05AFH  SYM  COMMANDTASK          00A7H  0051H  SYM  CONSOLECHAR

00A7H  003CH  SYM  COMMANDEXCEPTCOD     FC62H  06B5H  SYM  INITTASK  <-- INITIALIZATION TASK STARTING ADDRESS
                   -E

00A7H  003EH  SYM  INITEXCEPTCODE       FC62H  0084H  LIN    302
FC62H  0087H  LIN    304                FC62H  0093H  LIN    305
FC62H  0096H  LIN    306                FC62H  009DH  LIN    307
FC62H  00A1H  LIN    308                FC62H  00A4H  LIN    309
FC62H  00B0H  LIN    310                FC62H  00B3H  LIN    311
FC62H  00B9H  LIN    312                FC62H  00B9H  LIN    313
FC62H  00BCH  LIN    319                FC62H  00C2H  LIN    320
FC62H  00C8H  LIN    321                FC62H  00CEH  LIN    322
FC62H  00D1H  LIN    323                FC62H  00E4H  LIN    324
FC62H  00EFH  LIN    325                FC62H  00F8H  LIN    326
FC62H  010CH  LIN    327                FC62H  0116H  LIN    328
FC62H  011FH  LIN    329                FC62H  012CH  LIN    330
FC62H  0139H  LIN    331                FC62H  013BH  LIN    332
FC62H  013EH  LIN    335                FC62H  0143H  LIN    336
FC62H  0150H  LIN    337                FC62H  0150H  LIN    338
FC62H  015CH  LIN    339                FC62H  0160H  LIN    340
FC62H  016DH  LIN    341                FC62H  0170H  LIN    342
FC62H  0172H  LIN    344                FC62H  0175H  LIN    346
FC62H  017AH  LIN    349                FC62H  017FH  LIN    350
FC62H  0184H  LIN    351                FC62H  0189H  LIN    352
FC62H  0196H  LIN    353                FC62H  0196H  LIN    354
FC62H  01A5H  LIN    355                FC62H  01B0H  LIN    356
FC62H  01B0H  LIN    357                FC62H  01CDH  LIN    358
FC62H  01D6H  LIN    359                FC62H  01E6H  LIN    360
FC62H  01F5H  LIN    361                FC62H  0202H  LIN    362
FC62H  020FH  LIN    363                FC62H  021FH  LIN    364
FC62H  0228H  LIN    365                FC62H  0238H  LIN    366
FC62H  023BH  LIN    367                FC62H  0256H  LIN    369
FC62H  0259H  LIN    371                FC62H  0266H  LIN    372
FC62H  0270H  LIN    373                FC62H  0278H  LIN    375
FC62H  027DH  LIN    376                FC62H  028AH  LIN    377
FC62H  028DH  LIN    378                FC62H  029AH  LIN    379
FC62H  029CH  LIN    380                FC62H  02A0H  LIN    383
FC62H  02ACH  LIN    384                FC62H  02B8H  LIN    385
FC62H  02BBH  LIN    386                FC62H  02C2H  LIN    387
FC62H  02CAH  LIN    388                FC62H  02CFH  LIN    390
FC62H  02D2H  LIN    392                FC62H  02D7H  LIN    393
FC62H  02F3H  LIN    394                FC62H  0300H  LIN    395
FC62H  030FH  LIN    396                FC62H  031EH  LIN    397
FC62H  032DH  LIN    398                FC62H  033AH  LIN    399
FC62H  034FH  LIN    400                FC62H  0354H  LIN    401
FC62H  035DH  LIN    402                FC62H  0366H  LIN    403
FC62H  036FH  LIN    404                FC62H  037CH  LIN    405
FC62H  0389H  LIN    406                FC62H  0388H  LIN    407
FC62H  038EH  LIN    413                FC62H  039FH  LIN    416
FC62H  03A7H  LIN    417                FC62H  03ADH  LIN    418
FC62H  03BEH  LIN    419                FC62H  03D0H  LIN    420
FC62H  03D9H  LIN    421                FC62H  03F5H  LIN    422
FC62H  040EH  LIN    423                FC62H  0427H  LIN    424
FC62H  0440H  LIN    425                FC62H  0459H  LIN    426
FC62H  0472H  LIN    427                FC62H  0407H  LIN    428
FC62H  0489H  LIN    429                FC62H  04B9H  LIN    430
FC62H  049CH  LIN    439                FC62H  04DDH  LIN    440
FC62H  04A5H  LIN    441                FC62H  04A0H  LIN    442
FC62H  04BCH  LIN    443                FC62H  04CEH  LIN    444
FC62H  04D7H  LIN    445                FC62H  04DFH  LIN    446
FC62H  04EEH  LIN    447                FC62H  050BH  LIN    448
FC62H  050FH  LIN    449                FC62H  0518H  LIN    450
FC62H  052DH  LIN    451                FC62H  052FH  LIN    452
FC62H  0532H  LIN    460                FC62H  053FH  LIN    461
FC62H  053FH  LIN    462                FC62H  055AH  LIN    463
FC62H  0560H  LIN    464                FC62H  0568H  LIN    465
FC62H  0573H  LIN    466                FC62H  0588H  LIN    467
FC62H  0592H  LIN    468                FC62H  059DH  LIN    469
FC62H  05AAH  LIN    470                FC62H  05ADH  LIN    471
FC62H  05AFH  LIN    472                FC62H  05B2H  LIN    475
FC62H  05BFH  LIN    476                FC62H  05BFH  LIN    477
FC62H  05C9H  LIN    478                FC62H  05D0H  LIN    479
FC62H  05DAH  LIN    480                FC62H  05E0H  LIN    481
FC62H  05F4H  LIN    483                FC62H  05FAH  LIN    484
FC62H  0600H  LIN    485                FC62H  0610H  LIN    486
FC62H  0616H  LIN    487                FC62H  061CH  LIN    488
FC62H  062CH  LIN    489                FC62H  063CH  LIN    490
FC62H  064CH  LIN    491                FC62H  065CH  LIN    492
FC62H  066CH  LIN    493                FC62H  067CH  LIN    494
FC62H  068CH  LIN    495                FC62H  069CH  LIN    496
FC62H  06B0H  LIN    498                FC62H  06B3H  LIN    499
FC62H  06B5H  LIN    500                FC62H  06B8H  LIN    502
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
FC62H  06C4H  LIN    503              FC62H  06D5H  LIN    504
FC62H  06E6H  LIN    505              FC62H  06F6H  LIN    506
FC62H  071FH  LIN    507              FC62H  072CH  LIN    508
FC62H  0755H  LIN    509              FC62H  0762H  LIN    510
FC62H  078BH  LIN    511              FC62H  0798H  LIN    512
FC62H  07C1H  LIN    513              FC62H  07CEH  LIN    514
FC62H  07F7H  LIN    515              FC62H  0804H  LIN    516
FC62H  082DH  LIN    517              FC62H  083AH  LIN    518
FC62H  0863H  LIN    519              FC62H  0874H  LIN    520
FC62H  0084H  LIN    521

MEMORY MAP OF MODULE DEMO130
READ FROM FILE  F1.AP130.LNK
WRITTEN TO FILE F1.AP130

SEGMENT MAP

START     STOP      LENGTH ALIGN NAME            CLASS

00A70H    00AC1H    0052H  W   DATA              DATA      <-- LAST DATA BYTE OF APPLICATION JOB

00AC2H    00AC2H    0000H  W   STACK             STACK
00AD0H    00AD0H    0000H  B   ??SEG

FC620H    FD17BH    059CH  W   CODE              CODE      <-- LAST CODE BYTE OF APPLICATION JOB

FD17CH    FD17CH    0000H  W   MEMORY            MEMORY


GROUP MAP

ADDRESS   GROUP OR SEGMENT NAME
FC620H    CGROUP
          CODE
00A70H    DGROUP
          DATA
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX F
ROOT JOB LOCATE MAP

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
ISIS-II MCS-86 LOCATER, V1.2 INVOKED BY:
LOC86 :F1:RJB130.lnk                &
      TO :F1:RJB130                 &
      MAP PRINT(:f1:RJB130.mp2)     &
      OC(noli, nopl, nocm, nosb)    &
      PC(noli, pl, nocm, nosb)      &
      SEGSIZE(stack(0))             &
      ORDER(classes(data, stack, memory))     &
      ADDRESSES(classes(code(0FD180H),        &
                        data(00AD0H)))
WARNING 26   DECREASING SIZE OF SEGMENT
  SEGMENT   STACK

SYMBOL TABLE OF MODULE ROOT
READ FROM FILE  F1 RJB130 LNK
WRITTEN TO FILE  F1 RJB130

BASE    OFFSET TYPE SYMBOL              BASE    OFFSET TYPE SYMBOL

FD18H   0180H  PUB  NUC_INIT_ENTRY      FD18H   0184H  PUB  CODEDATA

FD18H   0011H  PUB  RQSTARTADDRESS  <-- ROOT JOB STARTING ADDRESS   FD18H  0010H  PUB  INTERROR

FD18H   0000H  PUB  CRASH              FD18H   002AH  PUB  RQROOTJOBVERSION
FD18H   0030H  PUB  ROOTTASK           FD18H   010CH  PUB  SYSTEMBUICIDE
FD18H   0118H  PUB  RQCREATEJOB        FD18H   011EH  PUB  RQGETTASKTOKENS
FD18H   0124H  PUB  RQSUSPENDTASK      FD18H   012AH  PUB  RQ_N_C_RETURN_6
FD18H   0146H  PUB  RQ_N_C_RETURN_40   FD18H   0162H  PUB  RQERROR
00ADH   0000H  PUB  JOBNUMBER          00ADH   0002H  PUB  ROOTTASKSTATUS

MEMORY MAP OF MODULE ROOT
READ FROM FILE  F1 RJB130 LNK
WRITTEN TO FILE  F1 RJB130

MODULE START ADDRESS  PARAGRAPH = FD18H  OFFSET = 0011H
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME            CLASS

00AD0H    00AD3H     0004H   W   DATA            DATA

00AD4H    00BFFH     012CH   W   INIT_STACK      STACK  <-- LAST DATA BYTE OF ROOT JOB

00C00H    00C00H     0000H   W   STACK           STACK
00C00H    00C00H     0000H   G   ??SEG
FD180H    FD339H     01BAH   W   CODE            CODE
FD33AH    FD345H     000CH   W   SAB_DESCRIPTOR  CODE
                               -8

FD346H    FD366H     0021H   W   U_J_DESCRIPTOR  CODE  <-- LAST CODE BYTE OF ROOT JOB
                               -9

FD36BH    FD36BH     0000H   W   MEMORY          MEMORY


GROUP MAP

ADDRESS   GROUP OR SEGMENT NAME
00AD0H    DGROUP
          DATA
FD180H    CGROUP
          CODE
          SAB_DESCRIPTORS
          U_J_DESCRIPTORS
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

APPLICATION NOTE

AP-174

January 1984

Optimizing the iRMX™ 86 Operating System Performance on System 86/310 and System 86/330

CATHERINE LUNDBERG
ISO APPLICATIONS MARKETING

© INTEL CORPORATION, 1984

Order Number: 230990-001

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Optimizing the iRMX™ 86 Operating System Performance on System 86/310 and System 86/330

CONTENTS

INTRODUCTION .....

OVERVIEW OF THE iRMX™ 86 OPERATING SYSTEM .....

PERFORMANCE TUNING .....
- The Size Of The Loader Buffers
- The Volume Granularity Of Devices
- The Number Of BIOS Buffers
- The Interleave Factor Of Devices

PERFORMANCE TESTS .....
- RSAT Maximum Transfer Rate
- iRMX™ 86 Operating System Generation
- iRMX™ 86 BIOS Generation
- COPY Test
- DIR Test
- Boot Disk Generation
- Boot Test
- BACKUP And RESTORE Test

EVALUATION SEQUENCE .....

SYSTEM 86/330A PERFORMANCE RESULTS .....

SYSTEM 86/310 PERFORMANCE RESULTS .....

CONCLUSION .....

APPENDIX A: SYSTEM CONFIGURATIONS .....

APPENDIX B: DEFINITION FILE FOR AN OPTIMUM iRMX™ 86 OPERATING SYSTEM .....

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INTRODUCTION

The Intel iRMX™ 86 Operating System is one of the most widely used real-time operating systems. Because it is intended for real-time applications it must respond immediately to the event that has the highest priority. Most users of the iRMX 86 Operating System are application program oriented and their programs use the real-time capabilities fully.

The Release 5 version of the iRMX 86 Operating System has the capability of supporting program development. This means that a system can be used for application code development, and later as the target system. Development costs can be decreased within a company, since there is no need to have separate systems for program development and for target systems. In addition, programs do not have to be downloaded from the development system to the target system.

Since there are many possible software configurations for mixes of real time applications and program development work, the purpose of this application note is to identify system configuration options which will improve the overall performance of the iRMX 86 Operating System from the Human Interface level, which is the level that the user sees while doing development work. The results of this application note could also be applicable to the user who is optimizing a target system configuration, since most of the parameters discussed in this study would affect the performance of the target system. Although the Release 5.0 operating system is faster and more optimized than previous versions of the iRMX 86 Operating System, further optimization has been found possible, especially when a specific hardware configuration is known.

OVERVIEW OF THE iRMX™ 86 OPERATING SYSTEM

The iRMX 86 Operating System is composed of layers. The layers in the iRMX 86 Release 5.0 Operating System are the Nucleus, the Basic I/O System, the Extended I/O System, the Application Loader, the Human Interface, and the Universal Development Interface (See Figure 1.) A layer provides a specific subset of operating system function. For instance, the Nucleus provides management of iRMX 86 objects such as tasks and jobs, the BIOS provides device independent I/O services, and the UDI provides the interface software to allow applications to be operating system independent.

Different layers are added according to the requirements of the application. Lower layers may be used without upper layers, but upper layers must have the lower layers as their groundwork. Using fewer layers of the operating system usually means that the application can execute faster. The UDI layer is required to run the utilities (such as PL/M 86, ASM86, and LINK86) since all the languages products are based on the UDI. The code produced by running these utilities may not need all the layers.

Most layers of the iRMX 86 Operating System have user configurable parameters. The values given to these parameters can be changed to allow the operating system to perform efficiently. Some parameters should be left at their default values, or the operating system will not function properly. For other parameters, the optimum value depends on the application, and the hardware configuration being used.

The System 86/310 and System 86/330A have different requirements because of their hardware implementations. The major difference is their Winchesters and flexible disks. Parameters that affect disk performance have the most effect on the systems. This application brief deals with those parameters which can be changed to provide optimal values for disk performance.

PERFORMANCE TUNING

A number of parameters can affect the system performance. The parameters which are configurable in the iRMX 86 Operating System don't change the actual speed at which the processor works. The variables that can improve performance affect how fast information can be given to the processor, or how much time must be spent to maintain system integrity. There are two ways to change the values of these parameters.

One way is to reconfigure the operating system using the ICU. The ICU is an interactive configuration utility which uses a screen oriented list of parameters to allow changes in an operating system. While using the ICU, different values are given to particular parameters to change the system performance. The ICU also allows the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

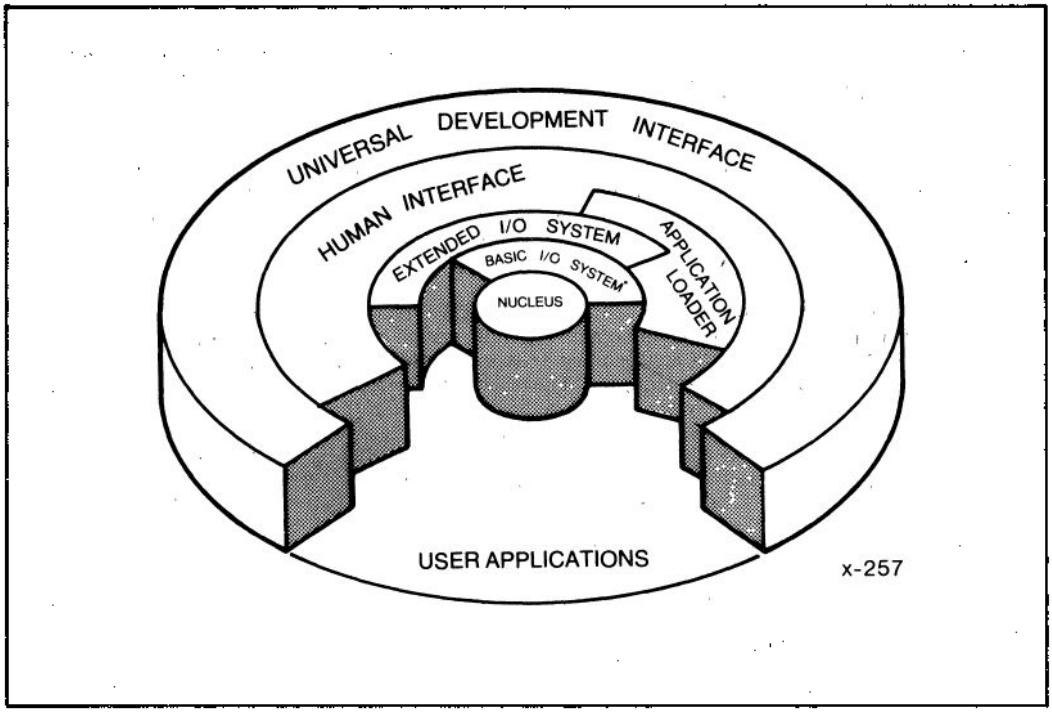


Figure 1 Model of the iRMX™86 Operating System

addition or deletion of individual layers of the operating system. The ICU creates a submit file from the information contained in the definition file. When this file is submitted, it generates a new operating system which uses the new values that were given to the parameters.

The second way parameters can be changed to affect system performance is to format the random access devices differently. Changing the interleave factor while using the FORMAT cusp makes a great difference in the disk I/O performance.

The Size of the Loader Buffers

The iRMX 86 Application Loader has two configurable buffer sizes. The Loader internal buffer is the Loader's working buffer for converting Object Module Formats (OMFs) to executable code. The default value for the Loader internal buffer is 400H, or 1K. Increasing this will not improve performance, but is sometimes necessary. OMFs are usually smaller than 1K, but the Fortran 86 compiler can sometimes generate records that are longer than 1K. Trying to load these records will cause an "E$REC_LENGTH" error, which can be eliminated by increasing the Loader internal buffer size. This parameter is found in the ICU screen titled "Application Loader". The variable is called "Internal Buffer Size (IBS)".

The Loader read buffer is used as a caching buffer when reading data from secondary storage. The Loader read buffer size influences the system performance. The Loader read buffer size is found in the ICU screen labeled "Application Loader". The parameter is labeled "Read Buffer Size (RBS)". The default value of the Loader read buffer size is 400H, or 1K. Increasing the Loader read buffer size will improve system performance.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Volume Granularity of Devices

File fragmentation happens when a file is written into a space that is not large enough to hold the whole file. Portions of the file must then be stored somewhere else on the disk to finish writing the file. This means that the disk controller must seek to several places on the disk to read the whole file, or to write the file, and the speed of disk I/O will be decreased. There are three different characteristics the user can vary to control file fragmentation on a device: device granularity, volume granularity, and file granularity. Device granularity is both the minimum allocation size, the minimum transfer size and is usually the sector size of a device. Volume granularity is the minimum file allocation size for all files on the device. The volume granularity is a multiple of the device granularity. Finally, file granularity is the minimum allocation size for a particular file and is a multiple of the volume granularity. By increasing the volume granularity and file granularity the files on a device should be forced to be contiguous. If the Winchester is fragmented, the smallest contiguous chunk of data for any one file will be the volume granularity.

Volume granularity on Winchester drives does not appear to affect system performance, so that parameter can be left at its default value of 400H, or 1K. This parameter is found in the screen titled "Intel iSBC® 215/iSBX 218 Device-Unit Information". The field is labeled "Granularity".

The diskette media which is used most frequently on the System 86/330A is a double sided double density disk with a device granularity of 256 bytes (100H). The physical name of the first such device is "wfdd0" in the standard definition file. With a device granularity of 256 bytes, a volume granularity of 256 bytes is the most effective. Using a device with a device granularity of 1024 bytes will improve the performance of the diskette. The physical name of the first double sided double density diskette with 1 K byte device granularity is "wfdx0". The device tested for this application note was "wfdd0" the diskette with device granularity of 256 bytes. The diskette device used in the System 86/310 has the physical name of "wmfdx0". It is a double sided, double density diskette with a device granularity of 512 bytes.

The Number of BIOS Buffers

The BIOS buffers are internal caching buffers which are used to hold data which is written to or read from secondary storage. The BIOS will only transfer the number of bytes that are a multiple of the device granularity. The device granularity is 1024 for the Winchester, and it varies for flexible disks. The most commonly used values are 512 for the System 86/310's diskette, and 256 for the System 86/330A's standard diskette.

Increasing the number of buffers for each device usually has a positive effect on disk performance, but only up to a certain point. Buffers must be updated, or flushed to the device, after a certain amount of time to ensure a reliable system. There are two parameters that control flushing buffers. They are called "Update Timeout," and "Fixed Update" or "Common Update Timeout". Update timeout is a variable for each device. A value of 64H (100) for update timeout means that when a device has not been accessed for 1 second, the buffers associated with that device will be written to the device. If the device is in constant use, this timeout limit may never be reached, especially if the value given to update timeout is large. For this reason, there is a second update parameter which can also be used.

The second parameter is called "Common Update Timeout" in the BIOS screen, and "Fixed Update" in the individual device screens. Common update timeout has one value for all devices in the operating system, but it is specified on an individual unit basis if it will be invoked. A value of 3E8H (1000) for common update timeout means that buffers associated with devices where common update timeout was invoked will be flushed at the end of 10 seconds. The buffers will be flushed if they have been used since the last time the system wide common update timeout happened. This ensures that even buffers for devices which have been continuously in use will be written to their device.

Increasing the amount of time between updating system buffers would improve performance, but it degrades reliability. If a system failure occurs, for instance, a power failure, the disks would have incorrect data on them if the buffers hadn't been written to disk. Using update timeout and common update timeout values that are appropriate reduces the damage caused by a system failure.

Different values of update timeout and common update timeout were not tested for the systems. These parameters were left at their default values for each device.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The time necessary to update the buffers causes a decrease in performance if there are too many buffers. Buffers also use memory when the device is connected. Careful usage of memory is necessary especially when the amount of memory available is limited, as is frequently the case for the target system.

Since diskettes are usually used only to transfer files between systems or for backups, the performance of the flexible disks is not as important as the performance of the Winchester device. Since increasing the number of buffers for a device increases the amount of memory required to attach the device, adding buffers for diskettes does not usually outweigh the need to conserve memory.

The BIOS number of buffers parameter is found in the ICU screen titled "Intel iSBC 215/iSBX 218 Device Unit Information". Each device has its own Device Unit Information screen. The field is labeled "Number of Buffers". The default number of buffers for Winchester devices is 4. The default number of buffers for flexible disk devices is usually 2.

The Interleave Factor of Devices

One of the parameters of the FORMAT command is "Interleave". If the consecutively-accessed sectors of a disk are staggered (that is, if they are not consecutive physical sectors), disk access time can decrease considerably. The reason for this decrease is that although a controller cannot read a sector and issue another read command in the time it takes for the next sector to be positioned under the head, the controller can perform this operation in less time than it takes for the disk to revolve once. Therefore, if the consecutively-accessed sectors are staggered correctly, the next accessed sector will be positioned under the read head just as the controller becomes ready to read it.

The amount of staggering is called the interleave factor. An interleave factor of two means that as the disk rotates, the controller consecutively accesses every second sector. Note that a properly set interleave factor also implies the number of disk rotations necessary to access all the sectors on a given track. An interleave factor of two implies that it takes two rotations of the disk to access all the sectors on a track.

The interleave factor is important when large transfers of consecutive data take place at speeds that approach the maximum transfer rate of the disk. Most information put on a disk will be stored in sectors that are consecutively accessed, if that is possible. If a disk has been heavily used so that few logically adjoining blocks are available, then the information will be stored in nonconsecutively accessed blocks, wherever there is space available. Naturally, this will slow down data transfer speed, since seeks must be done frequently to find where the next block of data is located. System performance will be best if the most frequently used utilities, programs and data are written onto the disk first, after formatting the disk with the optimum interleave factor.

There are three distinct cases where large amounts of data are transferred.

1) When the operating system is bootstrap loaded from disk

2) When the Application Loader is used to load an application program from disk

3) When programs are invoked that perform large transfers of consecutive data, such as the Human Interface COPY command

Each of these operations does a different amount of processing to the data which is being transferred. This means that the turnaround time between sector accesses is different.

The Bootstrap Loader instructs the disk controller to read one sector at a time. Thus, the turnaround time depends on the execution overhead of the Bootstrap Loader and is comparatively long. A large interleave factor is optimal for flexible disks that are used with the Bootstrap Loader. For hard disks however, the Bootstrap Loader has no effect on the turnaround time because revolution speed is so great that more than one disk revolution occurs between sector reads.

The Application Loader reads several sectors at a time into its internal read buffer. Then it takes a relatively long time to process the object records in this buffer. The ideal interleave factor here is one that optimizes for

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the object record processing time between disk accesses. For flexible diskettes, this interleave factor is somewhat smaller than that for the Bootstrap Loader. However, the Application Loader is not affected by the interleave factor on hard disks.

Applications which transfer large amounts of consecutive data (such as the COPY command) can initiate data transfers involving many sequential sectors. Thus, the controller accesses sectors on a given track as fast as possible. Here, the ideal interleave factor is one that optimizes for the turnaround speed of the disk controller.

The ideal interleave factor depends heavily on the application. However, because the revolution speed of hard disks is so high, they should be formatted with interleave factors that are optimized for the turnaround speed of the disk controller.

It is more important to match the interleave factor to the application with diskettes. They are usually smaller devices and are usually used for one major type of access. Flexible disks are much more sensitive to varying interleave factors, since the controllers for flexible devices are not as fast as the Winchester controllers. Different types of flexible disks will have different optimum values for the interleave factor. So optimum values for 8 inch diskettes will not be the same as optimum values for 5¼ inch diskettes.

The default value for the interleave factor in the FORMAT command is 5. The recommended Winchester interleave factor for the iRMX 86 Release 5.0 operating system was 4. This was evaluated to verify if this was the best interleave factor for the Winchester. Then flexible disks were evaluated to find out which interleave factor was best for each type of application.

PERFORMANCE TESTS

Since the goal of this application note was to determine optimum values of parameters when systems were being used for development, benchmarks which measure CPU performance were not used. Instead, all the tests used involved a lot of disk I/O. Some of them also involved building tables in memory. These tables could have been built on disk if there had been insufficient memory available for them, but that would have degraded performance markedly.

The languages and system utilities were used extensively, as well as the DIR, COPY, BACKUP and RESTORE cusps. These are the kinds of things that are done most frequently while using the system for development purposes. Descriptions of each of the tests used follow.

RSAT Maximum Transfer Rate:

The RSAT test (iRMX 86 System Acceptance Test) measures the number of bytes transferred every 60 seconds from a secondary storage device. It also keeps track of the maximum number of bytes transferred in a 60 second time period. If RSAT is invoked with a large buffer size, a good approximation of the maximum transfer rate of a device is obtained. A larger buffer size will increase the transfer rate. The largest buffer size that RSAT allows is 63K. This is a multiple of both the Winchester device granularity and the diskette device granularity, so it allows the maximum transfer rate.

The RSAT test is a composite of reads, writes, seeks, and truncates. The results are a good indication of overall performance during disk I/O. The RSAT test was run for both the Winchester and the diskette evaluations. The RSAT performance data contributed heavily in the analysis of the performance data.

iRMX™ 86 Operating System Generation:

The generation of an iRMX 86 Operating system from an Interactive Configuration Utility created submit file consists of a number of compilations and assemblies, and extensive use of the utilities. The generation of an iRMX 86 Operating System gives a very good indication of the performance that can be expected when using these utilities. The time necessary to complete the iRMX 86 Operating System generation was weighted heavily in the analysis of the performance data. This test involved I/O only on the Winchester.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

iRMX™ 86 BIOS Generation

Because the BIOS generation has fewer steps in it than a full iRMX 86 Operating System generation, it was used to zero in on the best configuration quickly. It assembles two modules, and then links two groups of libraries together. This test involved I/O only on the Winchester.

COPY Test:

The Human Interface COPY cusp was invoked to copy a large file (greater than 128K) from secondary storage. This test was used to see what the normal throughput of the system was. This was done for both Winchester and flexible disk devices.

DIR Test:

This test listed a directory with a large number of files in it to the terminal. The short file format was used to display the directory. This test was of some interest but was not weighted heavily in the analysis of the performance data. The time required to access a directory is extremely dependent on the location of the directory on the device. This test was done for the Winchester and flexible disk devices.

Boot Disk Generation:

The boot disk generation test formatted a diskette and then copied the files need for a bootable system onto the diskette. This test gave a good indication of the performance when writing to a diskette. This test was performed to determine diskette performance with different interleave factors.

Boot Test:

The amount of time it took the iRMX 86 Operating System to boot from a diskette was recorded to determine the best interleave factor for booting. This test was performed only on diskettes to determine the optimum interleave factor.

BACKUP And RESTORE Test:

The time to BACKUP and RESTORE from the Winchester to one flexible disk was gathered to determine the best interleave factor for the diskette when it is being used as a backup device. The iRMX 86 Release 5 versions of BACKUP and RESTORE were used in the evaluation. The volume granularity and BIOS buffer sizes were not a factor, since the diskette is formatted physical. BACKUP and RESTORE perform 1K reads and writes to the diskette. BACKUP and RESTORE use both Winchester I/O and diskette I/O, but the diskette was the only device which had the interleave factor changed as part of the evaluation with this test.

EVALUATION SEQUENCE

Since all possible permutations of variables could not be tested, the evaluation was performed by varying one parameter at a time. The research started with the default configurations of the operating system. The first parameter was varied to find the best value, and then its best value was used to determine the next parameter's optimum value. This method continued until all optimum values had been found. The parameters were tested in the following sequence.

1) The best Winchester interleave factor was found for the standard system. Values from 1 to 9 were tested for the System 86/330A, and interleave factors from 1 to 8 were tested for the System 86/310.

2) The best Application Loader read buffer size was determined. Buffer sizes from 1K to 8K bytes were tested for both systems in 1K byte increments.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3) The best number of Winchester BIOS buffers was determined. The systems were tested with 1 to 8 BIOS buffers for the Winchester.

4) The best Winchester volume granularity was found. Volume granularities of 1K and 2K bytes were tested for each system.

5) The best Winchester interleave factor was determined. Again, Winchester interleaves from 1 to 9 were tested for the System 86/330A, and Winchester interleaves from 1 to 8 were tested for the System 86/310.

6) The best number of BIOS buffers was determined for the diskettes. The systems were tested with 1 to 6 BIOS buffers.

7) The best diskette volume granularity was determined. Values of 256, 512 and 1024 bytes were tested for the System 86/330A diskette. Values of 512 and 1024 bytes were tested for the System 86/310 diskette.

8) The best flexible disk interleave factor was determined for each operation. Interleave factors from 1 to 8 were tested for the System 86/330A diskette. Interleave factors from 1 to 7 were tested for the System 86/310 diskette.

SYSTEM 86/330A PERFORMANCE RESULTS

Performance data was collected on a production System 86/330A using the iRMX 86 Release 5.0 Operating System in its standard configuration. Tests were run to determine the best configuration parameter values. Performance data was again collected with the best configuration of the iRMX 86 Operating System to determine the improvement in performance. The results for both configurations of the operating system using the 8" Priam Winchester using the iSBC 215 controller are shown in Tables 1 and 2 for the 5 MHz system and the 8 MHz system.

Table 1. System 86/330A Winchester Performance (5 MHz iSBC® 86/30 Single Board Computer)

Test	Execution Time		
	Standard (min:sec)	Optimum (min:sec)	Improvement (percent)
iRMX 86 Generation	17:22	16:08	7%
BIOS Generation	4:20	4:05	6%
DIR of 171 files	0:25	0:25	0%
COPY 128 K byte file	0:10	0:10	0%
	Bytes per Second		
	Standard	Optimum	Improvement
RSAT max transfer rate	80,640	90,316	12%

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 2. System 86/330A Winchester Performance (8 MHz iSBC® 86/30 Single Board Computer)

Test	Execution Time		
	Standard (min:sec)	Optimum (min:sec)	Improvement (percent)
iRMX 86 Generation	13:26	12:29	7%
BIOS Generation	3:21	3:09	6%
DIR of 171 files	0:22	0:19	14%
COPY 128 K byte file	0:10	0.08	20%
	Bytes per Second		
	Standard	Optimum	Improvement
RSAT max transfer rate	92,467	105,370	12%

Note that the granularity of the measurements was 1 second. In a test that takes 10 seconds to run, the real amount of time necessary to run a test could be 10% off of the result shown. This can account for the differences shown between the 5 and 8 MHz systems, especially in the DIR and COPY command tests. The tests which took greater quantities of time are a more accurate reflection of the actual system performance that can be expected.

The optimum values of the parameters are listed below. These were the parameter values which were used in the optimal iRMX 86 operating system configuration.

- The Application Loader read buffer size was increased from 1K to 7K.
- The volume granularity of the Winchester was left at 1K, which is the device granularity for the Winchester.
- The number of Winchester BIOS buffers was increased from 4 to 8.
- A Winchester interleave factor of 3 instead of 4 was used.
- The volume granularity of the diskette was left at 256 bytes. This was the 8" diskette's device granularity.
- The number of diskette BIOS buffers was increased from 2 to 4.

Performance data was collected on the 8" DS/DD diskette drive using the iSBX™ 218 controller after finding the optimum values of the parameters for the rest of the System 86/330A. The best interleave factor for the diskette was determined for each type of use. The results are shown in Table 3.

The fastest boot time was found when the diskette was formatted with an interleave factor of 7. For transferring files between systems by using the COPY cusp the interleave factor should be 3. This is shown in Table 4 by the results of the Boot Disk Generation test, the DIR test, and the COPY test. When treating an 8" diskette as a physical device, as in BACKUP and RESTORE, the interleave factor should be 2.

SYSTEM 86/310 PERFORMANCE RESULTS

Performance data was collected on a System 86/310 using the iRMX 86 Release 5.1 Operating System in its standard configuration. Tests were run to determine the best configuration parameter values. Performance data was again collected with the best configuration of the iRMX 86 Operating System to determine the improvement in performance. The result for both configurations of the operating system at 5 MHz using the 5¼" CMI Winchester with the iSBC 215 controller is shown in Table 5. The result for the optimum configuration of the operating system at 8 MHz using the 5¼" CMI Winchester is shown in Table 6.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 3. System 86/330A DS/DD Diskette Performance

Test	Execution Time (min:sec)		
Interleave Factor	**7**	**3**	**2**
Boot Time	1:02	2:42	2:36
Boot Disk Generation	6:34	6:18	6:43
DIR of 14 Files	0:22	0:21	0:22
COPY 187, 768 Byte File	1:01	0:45	1:03
BACKUP	4:15	2:43	2:25
RESTORE	4:40	2:43	2:25
	Bytes per Second		
RSAT Max Transfer Rate	5,376	9,677	12,902

Table 4. Optimum Interleave Factors for System 86/330A Diskettes

Type of Application	Optimum Interleave Factor
Using Bootstrap Loader	7
Transferring Named Files	3
BACKUP and RESTORE	2

Table 5. System 86/310 Winchester Performance (5MHz iSBC® 86/30 Single Board Computer)

Test	Execution Time		
	Standard (min:sec)	**Optimum (min:sec)**	**Improvement (percent)**
iRMX 86 Generation	19:16	17:57	7%
BIOS Generation	4:26	4:22	1%
DIR of 171 Files	0:27	0:24	11%
COPY 128 K Byte File	0:14	0:13	7%
	Bytes per Second		
	Standard	**Optimum**	**Improvement**
RSAT Max Transfer Rate	72,038	69,888	−3%

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 6. System 86/310 Winchester Performance (8 MHz iSBC® 86/30 Single Board Computer)

Test	Execution Time
	Optimum (min:sec)
iRMX 86 Generation	14:35
BIOS Generation	3:32
DIR of 171 Files	0:20
COPY 128 K Byte File	0:11
	Bytes per Second
	Optimum
RSAT Max Transfer Rate	78,490

The best performance of the System 86/310 was found with the following configuration of the parameters.

- The Application Loader read buffer size was increased from 1K to 7K bytes.
- The volume granularity of the Winchester was left at 1K bytes.
- The number of Winchester BIOS buffers was increased from 4 to 8.
- A Winchester interleave factor of 4 was used.
- The volume granularity of the diskette was left at 256 bytes.
- The number of diskette BIOS buffers was increased from 2 to 4.

After the optimum values were found for the variables affecting Winchester performance, performance data was collected on the 5.25" DS/DD diskette drive using the iSBX 218A controller to determine the best interleave factor for the diskette. Again, different interleave factors were best for different uses of the flexible disk. The results are shown in Table 7.

Table 7. System 86/310 DS/DD Diskette Performance

Test	Execution Time (min:sec)		
Interleave Factor	5	2	1
Boot Time	1:20	1:55	1:55
Boot Disk Generation	12:50	10:59	11:20
DIR of 14 Files	0:24	0:27	0:24
COPY 181,784 Byte File	1:17	1:01	1:16
BACKUP	2:02	1:20	1:10
RESTORE	2:04	1:11	1:00
	Bytes per Second		
RSAT Max Transfer Rate	3,226	6,451	9,677

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

As the data shows, the best interleave factor for booting was 5. For ordinary use in transferring files between systems with the COPY cusp the interleave factor should be 2. This was demonstrated by the Boot Disk Generation Test, the DIR test and the COPY test. For BACKUP and RESTORE from the Winchester to the flexible disk the best interleave factor was 1. These numbers are shown in Table 8 below.

Table 8. Optimum Interleave Factors for System 86/310 Diskettes

Type of Application	Optimum Interleave Factor
Using Bootstrap Loader	5
Transferring Named Files	2
BACKUP and RESTORE	1

CONCLUSION

The parameters changed generally affected I/O performance the most. The Application Loader read buffer size was changed from 1K to 7K bytes. The number of BIOS buffers was changed from 2 to 4 for flexible diskettes, and from 4 to 8 for Winchester devices. The interleave factor was set to 3 for the System 86/330A Winchester, and to 4 for the System 86/310 Winchester. The optimum interleave factor for each system's flexible diskette varied according to how the diskette was to be used.

By reconfiguring the system with different values for the configuration parameters and no changes to the hardware, a performance improvement of up to 20% may result. Performance may also be improved by changing the interleave factor when formatting random access disk devices. An application that used disk I/O would benefit by using the optimum values found in this application note.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX A
SYSTEM CONFIGURATIONS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX A

SYSTEM CONFIGURATIONS

Both systems used in the performance testing were production model systems. The software used on them was the iRMX 86 Release 5.0 Operating System for the System 86/330A, and the iRMX 86 Release 5.1 Operating System for the System 86/310. Hardware and software configurations of each system are shown in Table A-1.

Table A-1. System Configurations

System 86/330A	System 86/310
iSBC 86/30 Single Board Computer 384 K Memory iSBC 215/218 Disk Controller iRMX 86 Release 5.0 Operating System	iSBC 86/30 Single Board Computer 640 K Memory iSBC 215/218 Disk Controller iRMX 86 Release 5.1 Operating System

Note: More memory was required to run the ICU than was available in the System 86/330A. The ICU requires 448K bytes of RAM to run.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX B
DEFINITION FILE
OF AN OPTIMIZED iRMX™
86 OPERATING SYSTEM

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX B

Definition File of an Optimized iRMX™ 86 Operating System

Hardware

(OSP) 80130 Operating System Extension [Yes/No]	No
(OTU) 80130 Timer Used [Yes/No]	No
(OPU) 80130 PIC used [Yes/No]	No
(OCD) 80130 Copyright = 1981 [Yes/No]	Yes
(BL) 80130 Base Address Location [40h-0FFFFh]	0000H
(BP) 80130 Base Port Address [0-0FFFFH]	0000H
(MP) 8259A Master Port [0-0FFFFH]	00C0H
(MPS) Master PIC Port Separation [0-0FFH]	0002H
(SIL) Slave Interrupt Levels [1-7/None]	None
(LSS) Level Sensitive Slaves [1-7/None]	None
(LSP) Local slave PICS [1-7/None]	None
(TP) 8253 Timer Port [0-0FFFFH]	00D0H
(CIL) Clock Interrupt Level [0-7]	0002H
(CN) Timer Counter Number [0,1,2]	0000H
(CI) Clock Interval [0-0FFFFH msec]	000AH
(CF) Clock Frequency [0-0FFFFH khz]	04CDH
(TPS) Timer Port Separation [0-0FFH]	0002H
(NPX) Numeric Processor Extension [Yes/No]	Yes
(NIL) NPX Interrupt Level [Encoded]	0008H

Memory

Type : RAM = low, high
Type : ROM = low, high
Type : RAM = 0104H, 239FH
Type : RAM = 28CDH, F7FFH

Sub-systems

(UDI) Universal Development Interface [Yes/No]	Yes
(HI) Human Interface [Yes/No]	Req
(AL) Application Loader [Yes/No]	Req
(EIO) Extended I/O System [Yes/No]	Req
(BIO) Basic I/O System [Yes/No]	Req
(DB) Debugger [Yes/No]	No
(TH) Terminal Handler [Yes/No]	No
(CA) Crash Analyzer [Yes/No]	No
(UIR) UDI in ROM [Yes/No]	No
(CAR) Crash Analyzer in ROM [Yes/No]	No
(RIR) Root Job in ROM [Yes/No]	No

Human Interface

(ICL) Initial Command Line Size [0-0FFFFH]	0100H
(CNM) Command Name Length [0-255]	0030H
(SYS) System Directory [1-45 characters]	:SD:SYSTEM
(DRP) Default Resident Initial Program [Yes/No]	Yes
(RIP) Resident Initial Program [1-45 characters]	Default
(CDN) Configuration Device Name [1-14 chars]	:SD:
(PMI) Human Interface Pool Minimum [0-0FFFFH]	0100H
(PMA) Human Interface Pool Maximum [0-0FFFFH]	FFFFH
(HIR) Human Interface in ROM [Yes/No]	No

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

HI Jobs

(MIN) Jobs Minimum Memory [0-0FFFFH pages]	0100H
(MAX) Jobs Maximum Memory [0-0FFFFH pages]	0000H
(NPX) Numeric Processor Extension Used [Yes/No]	Yes

Resident User

(TDN) Terminal Device Name [1-12 characters]	TO
(MTP) Maximum Task Priority [0-0FFH]	00A0H
(UID) User ID Number [0-0FFFFH]	FFFFH
(MIN) Minimum Memory Required [0-0FFFFH]	0100H
(MAX) Maximum Memory Required [0-0FFFFH]	FFFFH
(IPP) Initial-Program Pathname [RESIDENT/1-45 characters]	RESIDENT
(DEF) Default Directory [1-45 characters]	:sd:user/world

Prefixes

Prefix: 1-45 characters
Prefix: :$:
Prefix: :PROG:
Prefix: :UTILS:
Prefix: :SYSTEM:
Prefix: :LANG:
Prefix:

HI Logical Names

Logical Name: logical_name,path_name [1-12 Chars, 1-45 Chars]
Logical Name: LANG, :SD:WORK
Logical Name: WORK, :SD:WORK
Logical Name: SYSTEM, :SD:SYSTEM
Logical Name: UTILS, :SD:UTILS

Application Loader

(IBS) Internal Buffer Size [0-0FFFFh]	0400H
(RBS) Read Buffer Size [0-0FFFFh]	1C00H
(LJT) Load Job Type [None/Async/Sync]	Synchronous and Asynchronous
(DMP) Default Memory Pool Size [0-0FFFFh]	0100H
(CT) Code Type [Abs/Pic/Ltl/Ovr]	Overlay, LTL, PIC and Abs
(ALR) Application Loader in ROM [Yes/No]	No

EIOS

(ASC) All Sys Calls in EIOS	Req
(ABR) Automatic Boot Device Recognition [Yes/No]	Yes
(DLN) Default System Device Logical Name [1-12 characters]	sd
(DPN) Default System Device Physical Name [1-12 characters]	wf0
(DFD) Default System Device File Driver [Phys/Str/Named]	Named
(DO) Default System Device Owners ID [0-0FFFFH]	0000H
(EBS) Internal Buffer Size [0-0FFFFh]	0400H
(DDS) Default IO Job Directory Size [5-0FF0h]	0020H
(ITP) Internal EIOS Task's Priorities [0-0FFH]	0083H
(PMI) EIOS Pool Minimum [0-0FFFFH]	0180H
(PMA) EIOS Pool Maximum [0-0FFFFH]	0180H
(EIR) Extended I/O System in ROM [Yes/No]	No

I/O Users

User: user name,Owner-ID [,ID,ID,ID,ID]

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Logical Names

Logical Name: logical name,device_name,file_driver,owners-id	[1-12 Chars ,1-14 Chars, Physical/Stream/Named, 0-0FFFFH]
Logical Name: BB, BB, Physical, 0000H	
Logical Name: STREAM,STREAM, Stream, 0000H	
Logical Name: LP,LP, Physical, 0000H	

BIOS

(ASC) All Sys Calls in BIOS [Yes/No]	Req
(ADP) Attach Device Task Priority [1-0FFH]	0081H
(TF) Timing Facilities Required [Yes/No]	Yes
(TTP) Timer Task Priority [0-0FFH]	0081H
(CON) Connection Job Delete Priority [0-0FFH]	0082H
(ACE) Ability to Create Existing Files [Yes/No]	Yes
(SMI) System Manager ID [Yes/No]	Yes
(CUT) Common Update Timeout [0-0FFFFH]	03E8H
(CST) Control-Sequence Translation [Yes/No]	Req
(PMI) BIOS Pool Minimum [0-0FFFFH]	0800H
(PMA) BOIS Pool Maximum [0-0FFFFH]	0800H
(BIR) Basic I/O System in ROM [Yes/No]	No

Intel Terminal Driver

(IIL) Input Interrupt Level [Encoded]	0068H
(OIL) Output Interrupt Level [Encoded]	0078H
(UDP) USART Data Port [0-0FFFFH]	00D8H
(USP) USART Status Port [0-0FFFFH]	00DAH
(IRP) 8253 Input Rate Port [0-0FFFFH]	00D4H
(ICP) 8253 Input Control Port [0-0FFFFH]	00D6H
(IRC) 8253 Input Counter Number [0-2]	0002H
(IRM) Input Rate Maximum [0-0FFFFFFFFH]	00012C00H
(ORP) 8253 Output Rate Port [0-0FFFFH]	0000H
(OCP)8253 Output Control Port [0-0FFFFH]	0000H
(ORC) 8253 Output Counter Number [0-2]	0000H
(ORM) Output Rate Maximum [0-0FFFFFFFFH]	00000000H

Intel Terminal Driver Unit Information

(NAM) Unit Info Name [1-17 Chars]	t0_info
(LEM) Line Edit Mode [Trans/Normal/Flush]	Normal
(ECH) Echo Mode [Yes/No]	Yes
(IPC) Input Parity Control [Yes/No]	Yes
(OPC) Output Parity Control [Yes/No]	Yes
(OCC) Output Control in Input [Yes/No]	Yes
(OSC) OSC Controls [Both/In/Out/Neither]	Both
(DUP) Duplex Mode [Full/Half]	Full
(TRM) Terminal Type [CRT/Hard Copy]	CRT
(MC) Modem Control [Yes/No]	No
(RPC) Read Parity Checking [See Help/0-3]	0000H
(WPC) Write Parity Checking [See Help/0-4]	0000H
(BR) Baud Rate [0-0FFFFH]	2580H
(SN)Scroll Number [0-0FFFFH]	0017H

Intel Terminal Driver Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	TO
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info Name [1-17 Chars]	t0_info
(MB) Max Buffers [0-0FFH]	0000H

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel iSBC® 215/218 Driver

(IL) Interrupt Level [Encoded Level]	0058H
(ITP) Interrupt Task Priority [0-0FFH]	0082H
(WIP) Wakeup I/O Port [0-0FFFFH]	0100H

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_215gen
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	0001H
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0001H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0000H
(NS) Number of Sectors/Track [0-0FFFFH]	000CH
(NAC) Number of Aux. Cylinders [0-0FFH]	0001H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_215w5
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NS) Number of Cylinders [0-0FFFFH]	0132H
(NFH) Numbers of Fixed Platters/Disk [0-0FFH]	0004H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0000H
(NS) Number of Sectors/Track [0-0FFFFH]	0009H
(NAC) Number of Aux. Cylinders [0-0FFH]	000AH
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BIT) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_215w
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	020DH
(NFH) Numbers of Fixed Platters/Disk [0-0FFH]	0005H
(NRH) Number of Remove Platters/Disk [0-0FFH]	000H
(NS) Number of Sectors/Track [0-0FFFFH]	000CH
(NAC) Number of Aux. Cylinders [0-0FFH]	000AH
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BIT) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_215pt
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFFH]	01D2H
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0003H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0000H
(NS) Number of Sectors/Track [0-0FFFFH]	000CH
(NAC) Number of Aux. Cylinders [0-0FFH]	0006H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	unifo_215f
(MR) Maximum Retries [0-0FFFFH]	0009H

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	004DH
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0000H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0001H
(NS) Number of Sectors/Track [0-0FFFFH]	001AH
(NAC) Number of Aux. Cylinders [0-0FFH]	0000H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_215fd
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	004DH
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0000H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0002H
(NS) Number of Sectors/Track [0-0FFFFH]	001AH
(NAC) Number of Aux Cylinders [0-0FFH]	0000H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	Yes

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_shugart96
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	0050H
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0000H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0002H
(NS) Number of Sectors/Track [0-0FFFFH]	0008H
(NAC) Number of Aux. Cylinders [0-0FFH]	0000H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	No

Intel iSBC® 215/218 Unit Information

(NAM) Unit Info Name [1-17 Chars]	uinfo_shugart48
(MR) Maximum Retries [0-0FFFFH]	0009H
(CS) Cylinder Size [0-0FFFFH]	0000H
(NC) Number of Cylinders [0-0FFFFH]	0028H
(NFH) Number of Fixed Platters/Disk [0-0FFH]	0000H
(NRH) Number of Remove Platters/Disk [0-0FFH]	0002H
(NS) Number of Sectors/Track [0-0FFFFH]	0008H
(NAC) Number of Aux. Cylinders [0-0FFH]	0000H
(SSN) Starting Sector Number [0-0FFFFFFFFH]	00000000H
(BTI) Bad Track Information [Yes/No]	No

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	cm0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	00A68000H
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info name [1-17 Chars]	uinfo_215w5
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0008H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	iw0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	01E2D000H
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info name [1-17 Chars]	uinfo_215w
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0008H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wmfdx0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0200H
(DSZ) Device Size [0-0FFFFFFFFH]	0004F800H
(UN) Unit Number on this Device [0-0FFH]	0008H
(UIN) Unit Info Name [1-17 Chars]	uinfo_shugart48
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wfdd0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0100H
(DSZ) Device Size [0-0FFFFFFFFH]	000F9700H
(UN) Unit Number on this Device [0-0FFH]	0008H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215fd
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wfd0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0100H
(DSZ) Device Size [0-0FFFFFFFFH]	0007C500H

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(UN) Unit Number on this Device [0-0FFH]	0008H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215f
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wf0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0080H
(DSZ) Device Size [0-0FFFFFFFFH]	0003E900H
(UN) Unit Number on this Device [0-0FFH]	0008H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215f
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	cm1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	00A68000H
(UN) Unit Number on this Device [0-0FFH]	0001H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215w5
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	iw1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	01E2D000H
(UN) Unit Number on this Device [0-0FFH]	0001H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215w
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wmfdx1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0200H
(DSZ) Device Size [0-0FFFFFFFFH]	0004F800H
(UN) Unit Number on this Device [0-0FFH]	0009H
(UIN) Unit Info Name [1-17 Chars]	uinfo_shugart48
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wfdd1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0100H
(DSZ) Device Size [0-0FFFFFFFFH]	000F9700H
(UN) Unit Number on this Device [0-0FFH]	0009H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215fd
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wfd1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0100H
(DSZ) Device Size [0-0FFFFFFFFH]	0007C500H
(UN) Unit Number on this Device [0-0FFH]	0009H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215f
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wf1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0080H
(DSZ) Device Size [0-0FFFFFFFFH]	0003E900H
(UN) Unit Number on this Device [0-0FFH]	0009H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215f

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wmfdy0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0200H
(DSZ) Device Size [0-0FFFFFFFFH]	0009F800H
(UN) Unit Number on this Device [0-0FFH]	0008H
(UIN) Unit Info Name [1-17 Chars]	uinfo_shugart96
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	wmfdy1
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Double
(SDS) Single or Double Sided Disks [Single/Double]	Double
(EFI) 8 or 5 Inch Disks [8/5]	5
(GRA) Granularity [0-0FFFFH]	0200H
(DSZ) Device Size [0-0FFFFFFFFH]	0009F800H
(UN) Unit Number on this Device [0-0FFH]	0009H
(UIN) Unit Info Name [1-17 Chars]	uinfo_shugart96
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0004H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	pw0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single
(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	0102C000H
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215pt
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0008H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel iSBC® 215/iSBX™ 218 Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	w0
(PFD) Physical File Driver Required [Yes/No]	Yes
(NFD) Named File Driver Required [Yes/No]	Yes
(SDD) Single or Double Density Disks [Single/Double]	Single

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(SDS) Single or Double Sided Disks [Single/Double]	Single
(EFI) 8 or 5 Inch Disks [8/5]	8
(GRA) Granularity [0-0FFFFH]	0400H
(DSZ) Device Size [0-0FFFFFFFFH]	00000400H
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info Name [1-17 Chars]	uinfo_215gen
(UDT) Update Timeout [0-0FFFFH]	0064H
(NB) Number of Buffers [nonrandom = 0/rand = 1-0FFFFH]	0008H
(FUP) Fixed Update [True/False]	True
(MB) Max Buffers [0-0FFH]	00FFH

Intel Line Printer Driver

(IL) Interrupt Level [Encoded Level]	0048H
(ITP) Interrupt Task Priority [0-0FFH]	0082H
(POA) 8255A Port A address [0-0FFFFH]	00C8H
(POB) 8255A Port B Address [0-0FFFFH]	00CAH
(POC) 8255A Port C Address [0-0FFFFH]	00CCH
(CON) 8255A Control Port Address [0-0FFFFH]	00CEH
(TAB) Printer Expanded Tabs [Yes/No]	Yes

Intel Line Printer Driver Device-Unit Information

(NAM) Device-Unit Name [1-13 Chars]	lp
(UN) Unit Number on this Device [0-0FFH]	0000H
(UIN) Unit Info Name [1-17 Chars]	NOT REQUIRED
(MB) Max Buffers [0-0FFH]	0000H

Nucleus

(ASC) All Sys Calls [Yes/No]	Req
(PV) Parameter Validation [Yes/No]	Yes
(ROD) Root Object Directory Size [0-0FF0h]	0020H
(MTS) Minimum Transfer Size [0-0FFFFH]	0040H
(DEH) Default Exception Handler [Yes/No/Deb/Use]	Yes
(NEH) Name of Ex Handler Object Module [1-32chs]	
(EM) Exception Mode [Never/Program/Environ/All]	Never
(NR) Nucleus in ROM [Yes/No]	No

User Jobs

(ODS) Object Directory Size [0-0FF0H]	0010H
(PMI) Pool Minimum [20H - 0FFFFH]	0100H
(PMA) Pool Maximum [20H - 0FFFFH]	0100H
(MOB) Maximum Objects [1-0FFFFH]	0010H
(MTK) Maximum Tasks [1-0FFFFH]	0010H
(MPR) Maximum Priority [0-0FFH]	0000H
(AEH) Address of Exception Handler [CS:IP]	0000H:0000H
(EM) Exception Mode [Never/Prog/Environ/All]	Never
(PV) Parameter Validation [Yes/No]	Yes
(TP) Task Priority [0-0FFH]	0082H
(TSA) Task Start Address [CS:IP]	23A0H:0000H
(DSB) Data Segment Base [0-0FFFFH]	0000H
(SSA) Stack Segment Address [SS:SP]	0000H:0000H
(SS) Stack Size [0-0FFFFH]	0100H
(NPX) Numeric Processor Extension Used [Yes/No]	No

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Includes and Libraries

Path Name [1-45 Characters]
(UDF) UDI Includes and Libs
    /rmx86/udi/
(HIF) Human Interface Includes and Libs
    /rmx86/hi/
(EIF) Extended I/O system Includes and Libs
    /rmx86/eios/
(ALF) Application Loader Includes and Libs
    /rmx86/loader/
(BIF) Basic I/O System Includes and Libs
    /rmx86/ios/
(THF) Terminal Handler and Debugger Includes and Libs
    /rmx86/th/
(NUF) Nucleus and Root Job Includes and Libs
    /rmx86/nucleus/
(ILF) Interface Libraries
    /rmx86/lib/
(CAF) Crash Analyzer Includes and Libs
    /rmx86/crash/
(DTF) Development Tools Path Names
    :lang:

Generate File Names

File Name [1-55 Characters]
(ROF) ROM Code File Name
    rmx86.rom
(RAF) RAM Code File Name
    rmx86

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel Related Publications

iRMX 86 Release 5 Operator's Manual (172764-001)
iRMX 86 Configuration Guide (9803126-05)
System 86/330A Overview Manual (144680-001)
iRMX 86 Documentation Addendum for Release 5 (146032-001)
iRMX 86 Basic I/O System Reference Manual (9803123-05)
iRMX 86 Loader Reference Manual (143318-002)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

intel

APPLICATION NOTE

AP-184

August 1984

Writing Device Drivers For XENIX* 86 and 286 — Task or Trivia?

MOHANDAS NAIR
APPLICATIONS MARKETING
INTEGRATED SYSTEMS OPERATION

© INTEL CORPORATION, 1984

Order Number 280041-001

* XENIX is a trademark of MICROSOFT Corp.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WRITING DEVICE DRIVERS FOR XENIX 86 AND 286 — TASK OR TRIVIA?

CONTENTS

1.0 INTRODUCTION .....

2.0 DEFINITIONS .....

3.0 COMPONENTS OF THE XENIX* I/O ENVIRONMENT .....
- 3.1 The process' view of the kernel .....
- 3.2 The kernel's view of the process .....
- 3.3 The kernel's view of the driver .....
- 3.4 The drivers view of the kernel .....
- 3.5 The driver's view of the device .....
- 3.6 Putting the components together .....

4.0 TYPES OF DEVICES .....
- 4.1 Block devices .....
- 4.2 Character devices .....
- 4.3 Combination devices .....

5.0 DEVICES VIEWED FROM THE USER INTERFACE .....

6.0 THE ENVIRONMENT .....
- 6.1 The u-area .....
- 6.2 Task-time execution vs. interrupt-time execution .....
- 6.3 I/O path through the system ..

7.0 THE ANATOMY OF THE I/O SYSTEM .....
- 7.1 The block interface .....
  - 7.1.1 The system buffers .....
  - 7.1.2 Driver support routines ....
  - 7.1.3 Block device driver routines .....
  - 7.1.4 Review .....
  - 7.1.5 Steps taken to satisfy requests .....
  - 7.1.6 iSBC® 254 Bubble Memory board walkthrough .....
  - 7.1.7 Final look at block drivers .....
  - 7.1.8 The raw (character interface) to a block device .....
- 7.2 The character interface .....
  - 7.2.1 Clists .....
  - 7.2.2 Terminal I/O .....
  - 7.2.3 Useful routines .....
  - 7.2.4 tty.help — the line discipline routines .....
  - 7.2.5 Terminal I/O device driver routines .....
  - 7.2.6 Examples of character I/O (terminal) drivers .....
    - 7.2.6.1 ISBX™ 270 Walkthrough .....
    - 7.2.6.2 Low level routines .....
    - 7.2.6.3 Required routines .....
    - 7.2.6.4 A final note .....

CONCLUSIONS .....

APPENDICES .....

REFERENCES .....

ACKNOWLEDGEMENTS .....

* XENIX is a trademark of MICROSOFT Corp.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1.0 INTRODUCTION

The world of device configuration and device drivers has since time been an area where only hacks could tread. Being the lowest-level of software interfacing to the device, drivers were always examined by self-motivated experts. Also, drivers were hard to come-by and even harder to comprehend.

Intel Corporation's "open systems" concept, coupled with the XENIX* Operating System and our family of microprocessor systems, creates an attractive environment for building and adding new devices and drivers. However, the folklore involved with the XENIX Operating System and its internal functions are exemplified in the lack of device driver details. This paper clears the fog around device drivers and device driver writing. It is written for the general operating system user bringing him/her details of the anatomy of the I/O system, driver interfaces and driver support routines. The details that follow pertain to the Release 1 XENIX 286/86 Operating System which is a superset of the Unix* V7 operating system. This application note discusses writing device drivers for the XENIX Operating System as well as describes the operating environment around device drivers. Note that the reader may not need the discussion on the environment when writing device drivers. However, when the reader begins to debug them, he/she will find these discussions worthwhile.

Most driver writers would agree that few start writing drivers from scratch. Many use existing driver code as templates. This paper includes actual coded and pseudo-coded examples coupled with descriptions which grant the reader(and soon to be writer) a bouncing-board introduction to writing device drivers.

2.0 DEFINITIONS

A device driver is that body of software that allows an operating system to communicate with a device. This body of software is the lowest software level of abstraction in the I/O system. Figure 1a shows these levels identifying device drivers as a set of machine-independent routines(commands) of the operating system which talk to devices.

In the XENIX Operating System, a device driver is a collection of procedures placed in a file that is configured into the system. No source code is needed for this configuration as the operating system, once configured, will talk with these routines in the driver.

Before attempting to write a XENIX device driver, one must:

1) Understand the device i.e. know how to talk to the device, initialize it etc.
2) Understand how the XENIX Operating interfaces to the driver i.e what is covered in this application note.
3) Begin writing the routines needed for a driver i.e. mimic an existing driver.

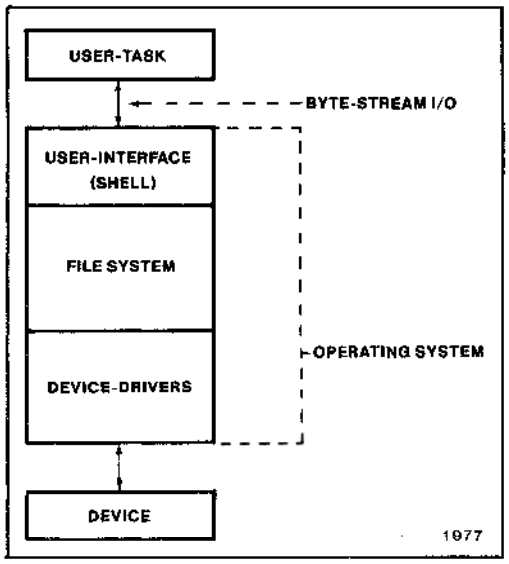


Figure 1a. Driver Model

3.0 COMPONENTS OF THE XENIX I/O ENVIRONMENT

This application note breaks device drivers and the XENIX Operating system into the following components:

1) the process
2) the kernel
3) the I/O buffers
4) the driver(s)
5) the device(s)

These components are shown in figure 1b. These components communicate with each other in unique ways.

3.1 The Process' View of the Kernel

Processes communicate with the kernel through system calls i.e. open, close, read. These system calls can be found in the XENIX Operating System Documentation (173258001).

3.2 The Kernel's View of the Process

Section 6.0 (THE ENVIRONMENT) defines a process, process synchronization, user-areas and introduces the kernel's view of the process.

*UNIX is a Trademark of Bell Labs

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

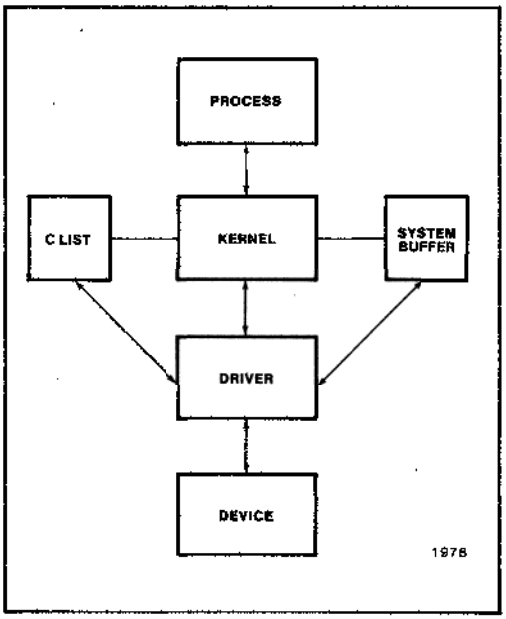


Figure 1b. Components of the I/O Environment

3.3 The Kernel's View of the Driver

Section 4.0 (Types of Devices) serves as an introduction to how the kernel views the driver. Details of this view are found in Section 7.0 (The Anatomy of the I/O System) where the system I/O buffers are described as used by the kernel.

3.4 The Drivers View of the Kernel

Section 7.0 (The Anatomy of the I/O System) explains how the driver communicates with the kernel, by introducing the assist routines (Section 7.1.2, 7.2.3, 7.2.4) available to drivers. Furthermore, the driver must understand the buffer scheme when talking with the kernel. Sections 7.1.1 and 7.2.1 (The I/O buffers) detail what the driver manipulates.

3.5 The Driver's View of the Device

The device driver is a collection of routines that act on the device and the device reponds to the driver through interrupts. The driver talks with the device when an action is requested or when the device interrupts the driver on completion of the action. The driver talks with the device through routines which are discussed in Sections 7.1.3 (Block device driver routines), 7.2.5 (Terminal I/O device driver routines and Appendix D (Interrupt Mapping).

3.6 Putting the Components Together

Section 6.3 (I/O Path through the system) will give an introduction to how the above-mentioned components interact. To understand this discussion, the following concepts must be covered:

1) Types of Devices (Section 4.0)
2) How Devices are viewed from the user interface (Section 5.0)
3) The kernel's view of the process (The Environment Section 6.0, 6.1 and 6.2)

4.0 TYPES OF DEVICES

The XENIX Operating System supports two kinds of devices - Block and Character. Input/Output to/from these devices are consequently known as block and character I/O.

4.1 Block Devices

A block device is a sectored device that is accessed randomly. A file system resides on it as well. A good example of a block device is a winchester disk or a floppy disk. I/O with a block device is executed through a set of kernel I/O buffers(cache) which intervenes transfers of data (in fixed sized blocks) between user memory and the respective device. Block I/O involves a considerable amount of kernel activity due to these buffering characteristics.

Also:

1) The size of Block I/O transfer requests from kernel to device are a multiple of the system's blocksize (BSIZE). BSIZE is 1024 bytes in XENIX 286 Operating System.
2) Transfers are seldom done directly to the user task's memory. The transfers are staged through a buffer pool of BSIZE buffers. Also, the XENIX kernel manages these buffers to perform blocking/deblocking and cacheing. I/O transfers to/from the user task's memory are satisfied from the buffers.

4.2 Character Devices

A character device is unstructured. A file-system cannot reside on a character device. Examples are terminals and printers. In character I/O, data transfer requests occur in 'n' bytes between sections of memory and the device. Hence, "character" is synonymous to "byte." One must realize that there is minimal operating system involvement in data transfer as it is a private transaction between a user task and the device driver(see figure 2).

4.3 Combination Devices

Some devices can be accessed and treated as a block or character device. For example, the disk interface can be accessed either as a block or character I/O

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

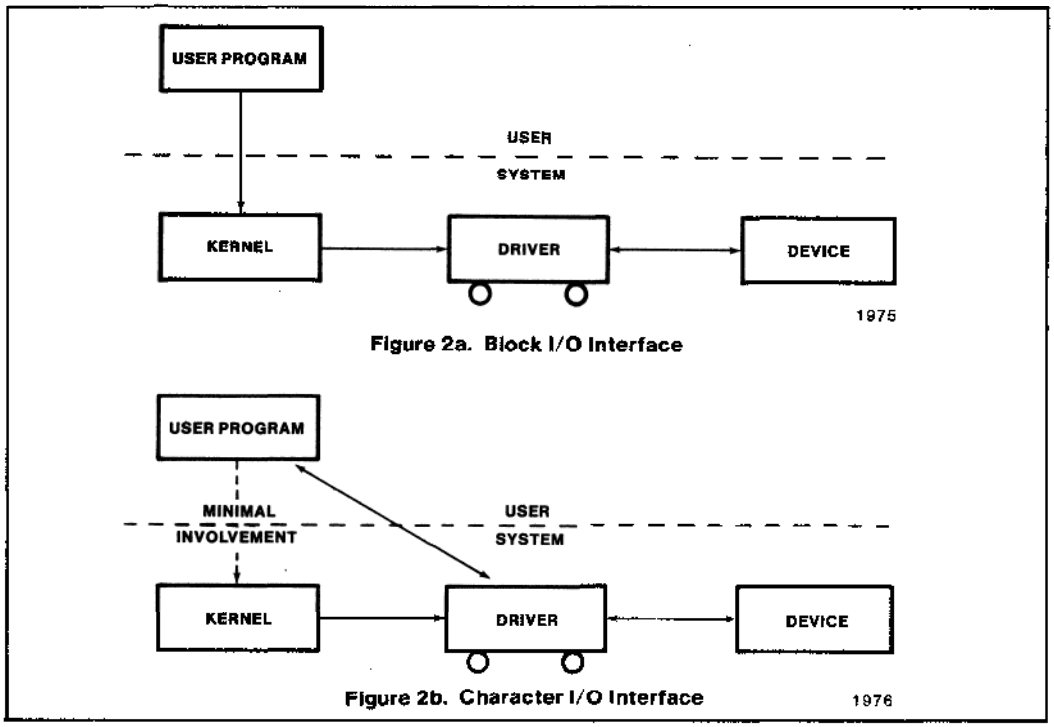


Figure 2a and 2b. Block and Character Interfaces

device. The character device permits direct i/o transfers between user memory and device. This mode of transfer is called RAW I/O. Raw I/O is very useful when direct disk-to-disk copy is necessary. The kernel buffers and file system are bypassed in this operation. Routines such as **dump, dd, fsck** are examples of such raw operations.

5.0 DEVICES VIEWED FROM THE USER INTERFACE

How are drivers in the XENIX Operating System identified? In the XENIX user interface, a directory called /dev exists for the purpose of holding all relevant device driver interfaces. XENIX is a file-oriented operating system and treats all devices as files. As files are accessible by the file-system, so are devices.

An **ls -l** (directory listing) of the **/dev** directory is shown in figure 3.

Since devices are accessible as files, data can be sent to them with:

```
echo "talk to me" > /dev/ttya0
```

Devices can be opened, updated and closed via the filesystem. **/dev/ttya0** is some-user's terminal. Files that identify devices are called device special files as they provide the hook to the drivers from the file-system. Enforcing device independence where all devices are files in XENIX permits tremendous flexibility and uniformity in the XENIX Operating System. Device special files are identified to the kernel as a 16-bit integer value. This 16-bit value is composed of two other values, which are the device's major and minor numbers. The high-order 8-bits form the major number and the low-order form the minor number. XENIX provides two macros — **major(dev)/minor(dev)** to decode these values from the device file. This <major,minor> number pair is used, whenever the device is referenced, to identify the relevant device driver. XENIX is internally very table(array) oriented. These numbers are indices to an array of possible drivers.

Figure 3 shows the <major,minor> number-pair for each device. Also note that the leading "c" and "b" characters first on each line denote the character and block files respectively. To see the connection between major number/minor number and drivers, take a look at the **c.c** file (appendix A). **c.c** is created in the configuration process. The following data-structures in **c.c** are created to maintain the relationship between drivers and the major/minor numbers

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

(identifying the device special file). The data structures are:

1) **dinitsw[ ]** is a vector of device-initialization procedures. The procedures mentioned in this vector are called during system initialization to initialize devices.

2) **bdevsw[ ]** is the table of block-device interfaces. The index to a driver in this table is the major device number of the Block interface for the device.

3) **cdevsw[ ]** is the table of character-device interfaces. The index to the driver in this table is the major device number of the character interface to the device. Note that major numbers do not overlap for block/character devices unless the same device is represented as a block and a character device.

4) **vecintsw[ ]** is the table of interrupt procedures. This table contains one entry per supported interrupt level of the cascaded 8259A Programmable Interrupt Controller. The index represents the interrupt level.

```
total 11
crw--w--w- 1 root     4,   1 Feb  1 13:13 console
brw-------  1 root     0,   9 Jan 23 17:47 df0
brw-------  1 root     0,  13 Jan 20 09:47 dxf0
brw-------  1 root     0,   8 Jan 20 09:46 f0
crw-------  1sysinfo   2,   1 Jan 20 09:45 kmem
crw--w--w- 1 root     3,   7 Feb  1 14:24 lp
crw-------  1 sysinfo  2,   0 Jan 20 09:45 mem
crw-rw-rw- 1 root     2,   2 Feb  1 19:30 null
crw-------  1 root     0,   9 Feb  1 07:32 rdf0
crw-------  1 root     0,  13 Jan 20 09:47 rdxf0
brw-------  2 sysinfo  0,   1 Jan 30 16:40 root
crw-------  2 sysinfo  0,   1 Jan 20 09:45 rroot
crw-------  2 sysinfo  0,   3 Jan 30 16:42 rusr
crw-------  2 sysinfo  0,   1 Jan 20 09:45 rw0a
crw-------  2 root     0,   2 Jan 20 09:45 rw0b
crw-------  2 sysinfo  0,   3 Jan 30 16:42 rw0c
crw-------  1 root     0,   0 Jan 20 09:45 rw0t0
crw-------  1 root     0,  12 Jan 20 09:47 rxf0
crw-rw-rw-  1 root     6,   0 Jan 20 09:45 tty
crw-w--w-  1 root     3,   0 Feb  1 17:35 ttya0
crw--w--w- 1 nair     3,   1 Feb  1 19:36 ttya1
crw-rw-rw- 1 root     3,   2 Feb  1 08:26 ttya2
crw--w--w- 1 root     3,   3 Feb  1 07:53 ttya3
crw-rw-rw- 1 root     4,   0 Jan 20 09:45 ttyf0
brw-------  2 sysinfo  0,   3 Jan 24 16:51 usr
brw-------  2 sysinfo  0,   1 Jan 30 16:40 w0a
brw-------  2 sysinfo  0,   2 Jan 20 09:45 w0b
brw-------  2 sysinfo  0,   3 Jan 24 16:51 w0c
brw-------  1 root     0,   0 Jan 20 09:45 w0t0
```

Figure 3. The ls Listing

Only an overview of configuration details will be covered in this write-up. Knowledge of the configuration process is not needed to write a device driver. Figure 4 illustrates the file and directory structure maintaing drivers and the configuration files/shell scripts. Basically, there are three interesting files:

1) xenixconf: edit this file to describe the configuration to be built (see Appendix B)

2) master: contains a master copy of the configuration information (Appendix C)

3) c.c: generated from the above two using a program "config."

config
i.e master + xenixconf ==========> c.c

To configure:

1) edit master

2) edit xenixconf

3) in **/sys/conf** run MAKEXENIX

The rest is automatic. Appendix B and C give examples of xenixconf and master. There will be no further discussion of the configuration details which do not contribute to learning how to write device drivers in XENIX. This overview was meant to describe how **c.c** is created. However, before attempting to write drivers, one must have a minimal understanding of the operating environment surrounding device drivers. This is the topic of discussion that follows.

6.0 THE ENVIRONMENT

The environment of any driver is, of course, the operating system, the device and the user-programs(processes) that communicate with the driver. This section concentrates on the kernel's view of the process. A task or process to the operating system is defined as:

1) the existence of an element/structure in the **proc table**. The proc table contains details of all processes in the system.

2) the existence of a **per-process data area (u-area)** representing it in the kernel. Any process image contains this special area that is copied into the kernel data space when it is active. This area identifies the process and hold parameters of the process.

A proc table entry and its corresponding **u_structure**, defines the state of a process at any instance during its birth(fork), lifetime and death(exit). The proc table entry is that part of a process that must always remain in memory for process communication and restart capability. The **u_structure** is that part of the process which can be swapped out onto disk, along with the per-process data segment, at times when the process is not in a runnable state.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

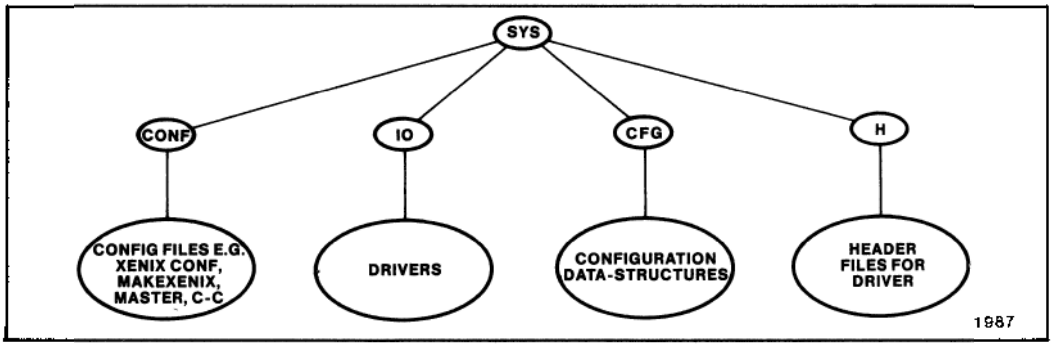


Figure 4. The Configuration Files/Directories

6.1 The u-area

The XENIX operating system does not differentiate a kernel process from a user process. Processes can run in either kernel mode, using system services and privileges (access to i/o drivers, u_area) or in user mode, where user-program code is executing. The operating system manages the relevant user-programs using a per-process data area, called the **u-area**. The **u-area** contains pertinent information such as system-stack information, preserved registers and I/O parameters used for data transfer. Throughout the discussion on device drivers, there will be mention of information in the **u-area**. To recap, the u-area is that space held in the kernel to maintain information about processes that run on the CPU. Every process has a u-area that is made up of a detailed structure, called the **u-structure**. This structure is multiplexed in the kernel i.e is swapped in and out with the process and contains considerable information about the process such as:

system call arguments
process sizes
registers saved
error information
I/O information

Input/output parameters held here are:

u.u_error -error information, 0 means no error
u.u_base - starting user transfer address
u.u_count - bytes to transfer
u.u_segflg - flag telling if transfer is to/from user data space (0) or system memory (1)
u.u_offset - offset in file for I/O

The driver will receive other information too, such as, the target device, the size of the job and the buffer address in the task's memory. For block devices, only **u.u_error** is updated. For character drivers, all other parameters are used. The u-area should not be accessed by the running driver at all times. This is because events occur in any operating system that are either:

1) Synchronous, as in normal code executing in user space
2) Asynchronous, as when an interrupt occurs and the interrupt service routine is called for the device.

Synchronous activities happen as the CPU permits users to run their code. When their code makes requests to the system resources, they are subject to be swapped out of memory. Once the device performs these requests, it interrupts the processor asynchronously. The Operating System then calls on certain routines, called interrupt service, routines, to perform actions that follow the device's completion of its requested job.

The **u_area**, being part of the process, may have been swapped out after resource requests were made. Hence, the asynchronous portions of code called on an interrupt cannot access this area. The terms "task-time execution" and "interrupt-time execution" are used to differentiate times it can and cannot be accessed. Obviously, the device driver must contain routines that are called due to asynchronous and synchronous events.

6.2 Task-time Execution vs. Interrupt-time Execution

As mentioned, the XENIX Operating System does not concern itself with whether tasks are running in user or kernel mode. Hence, there may be several tasks contending for system resources that are non-I/O related. At task-time, tasks are executing user or system code. Their **u-area** may be used whenever the process executes system code i.e. makes a system call and the kernel uses this area for stack and parameter storage. This **u-area** is resident when the process is running and ,therefore, can be addressed by the driver for data transfer. The user space is addressable at this time and information about the running process can be placed in this area by the driver.

Contrary to this, during interrupt-time execution, the device has interrupted the CPU. At this time, the running task may not be the task that requested

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I/O to/from the device. Usually, the task-time portion of the driver has exited after the I/O request and that process may have been swapped out. Hence, the **u-area** and the user data space cannot safely be accessed or used at interrupt time by the interrupt service routine. However, the interrupt service routine gets relevant information(about what to do) from the task-time portion of the driver via static variables. Obviously, from the nature of task-management, interrupt routines are limited in behavior and they cannot make assumptions about the state of the system or the presense of tasks/data in the system. Figure 5a illustrates the distinction between the task-time portion and the interrupt-time portion of any driver that has to evidently cater and respect the realities of task vs. interrupt -time execution.

Task and interrupt-time routines are separate but they may call common data-manipulating routines. One such routine is called by both sides and manipulates common buffers used (will be discussed later). There is a mechanism used for mutually excluding an update attempt of one routine from another. This mutual exclusion technique is required as the asynchronous portion of the driver may be removing an element from a list, for example, while the synchronous portion of the same driver may be placing an element onto the list. The technique basically deals with the interrupt structure and priorities of the system. For further details on the interrupt mapping, see Interrupt Mapping(Appendix D).

The operating environment, and the nature of tasks have been discussed only briefly as the device driver is the main focus of this application note. To bring together the basic components of the roadmap (Section 3.0) i.e. how the process, the kernel and the driver communicate, a brief description of the basic I/O path through the system follows.

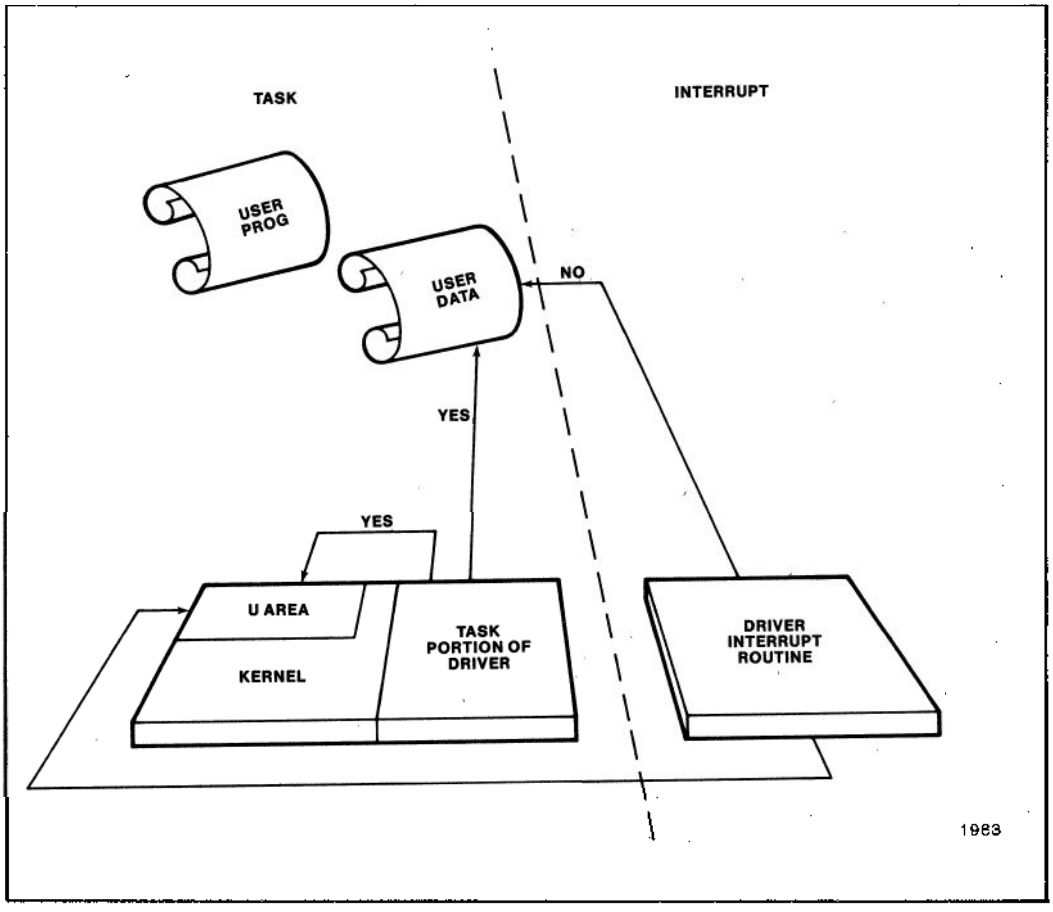


Figure 5a. Task vs. Interrupt Time Execution

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.3 I/O Path Through the System

Thus far, the driver interfaces have yet to be discussed. Knowing the environment a driver exists in is sufficient for understanding the I/O path through the kernel when I/O system calls are made. This will explain how the components of the roadmap (Section 3.0) interact. To illustrate the way all these device tables and system calls relate, the following calling sequences are presented. Details have been avoided to simplify the concepts described.

```
/*
* user program makes system call
*/

fd = open("/dev/ttyb1",1)

/*
* the kernel reacts with
*/
```

1) The kernel entry routine gets the system-call trap, determines it's an "open" call and calls "open" in the kernel.
2) Open calls a procedure to parse the path-name (nami) and turn it into an inode. An inode is the structure that represents a file in the file-system.
3) Open notices that the inode represents a character special-file and not a normal disk file. Thus, it accessed the **cdevsw[]** table using the major number in the inode, and calls the device-drivers open procedure.
4) The driver's open procedure does whatever it needs to do i.e. opens the device. It sets **u.u_error** if an error occurs and returns.
5) If the driver did not declare an error, open allocates a file-descriptor in the user-process, and returns it.
6) If the driver returned an error, open simply passes it back to the user program.

```
/*
* user program does
*/

read(fd, buf, 128);

/*
* kernel does
*/
```

1) The kernel gets the system-call trap, determines it is a read call and calls read in the kernel.
2) Read uses the "fd" argument and determines which inode it represents.
3) The inode indicates it is a character special-file. Thus, read uses the major device number stored in the inode to access the **cdevsw[]** table and calls the drivers read routine.
4) The drivers read routine does whatever it needs to do transferring data to the user's buffer. If any errors occur, it sets **u.u_error**. In any case, the driver eventually returns after having transferred the data.
5) The read routine returns to the kernel, which returns to the user program.

The I/O path will be elaborated on with details on driver functions with respect to block I/O. For the time being, the above sequence should be used as a template.

7.0 THE ANATOMY OF THE I/O SYSTEM

This discussion covers how the kernel and the driver communicate (See Section 3.0 Roadmap) and can be best broken down into two stages. They are:

1) the block interface
2) the character interface

The following discussions go into much detail on how the buffer schemes for block and character I/O devices work. These details are not needed for device driver writing. However, knowledge of the kernel buffers for block I/O devices and the character lists for character I/O devices will be useful when debugging begins. This knowledge will help the device driver writer to understand the inner workings of the driver he/she is writing.

7.1 The Block Interface

Much has been discussed about what block I/O is. Block I/O transfers require the kernel's intervention when they occur. All block transfers require the use of I/O buffers. These buffers are used as a temporary storage area for caching and blocking/deblocking. Usually, data transfer occurs between user space and devices via these system buffers (see figure 5b).

Each buffer is BSIZE bytes long and has a buffer header corresponding to it. BSIZE and other system level constants are defined in **param.h** (Appendix E) The header file **buf.h** defines this buffer header structure (Appendix F). **Buf.h** also describes how these kernel buffers are structured. Although it is not necessary to understand all fields in the buffer header, some fields must be noted:

b_dev- device number
b_blkno- block for the transfer
b_bcount- bytes to transfer

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

b_cylin- cylinder number
b_addr- address to be transferred from/to
b_flags- nature of transfer i.e. B_READ

7.1.1 THE SYSTEM BUFFERS

Consider a list, called the freelist, that is initialized to be a circular doubly-linked list of buffer-headers. Each header in the list has a pointer to its own BSIZE buffer. Figure 6a shows this list with forward and backward pointers **av_forw** and **av_back**. Consider another list called the device list. This list hangs off each device driver. The buffers forming these device lists are device request that are waiting to be serviced. At start state these lists for each device are empty and their headers point to themselves with **b_forw** and **b_back**. (figure 6b). Each device has its own queue structure that exists to schedule I/O requests. This queue is structured exactly like the freelist, i.e. is doubly linked and circular. The head of this queue called the static buffer header, is a buffer-header just like all the others. However, some irrelevant fields that it holds (used by other headers in the queue) are redeclared(aliased). These redeclarations are found in **buf.h**. Here, **av_forw**, **av_back**, **b_forw**, **b_back** used on the same buffer headers but form two concurrent lists. Remember, the freelist is the master list. **b_flags** determines, for the enquirer, if the buffer is BUSY, WAITING and the like. Details of these flags are found in **buf.h** also.(for further details about flags see ref 1).

The configuration file c.c (appendix A) holds the data structure **bdevsw[ ]** which is the table of device driver routines indexed by their major number. The last element in this structure for each device is ixxxtab (see naming convention — Appendix G) where

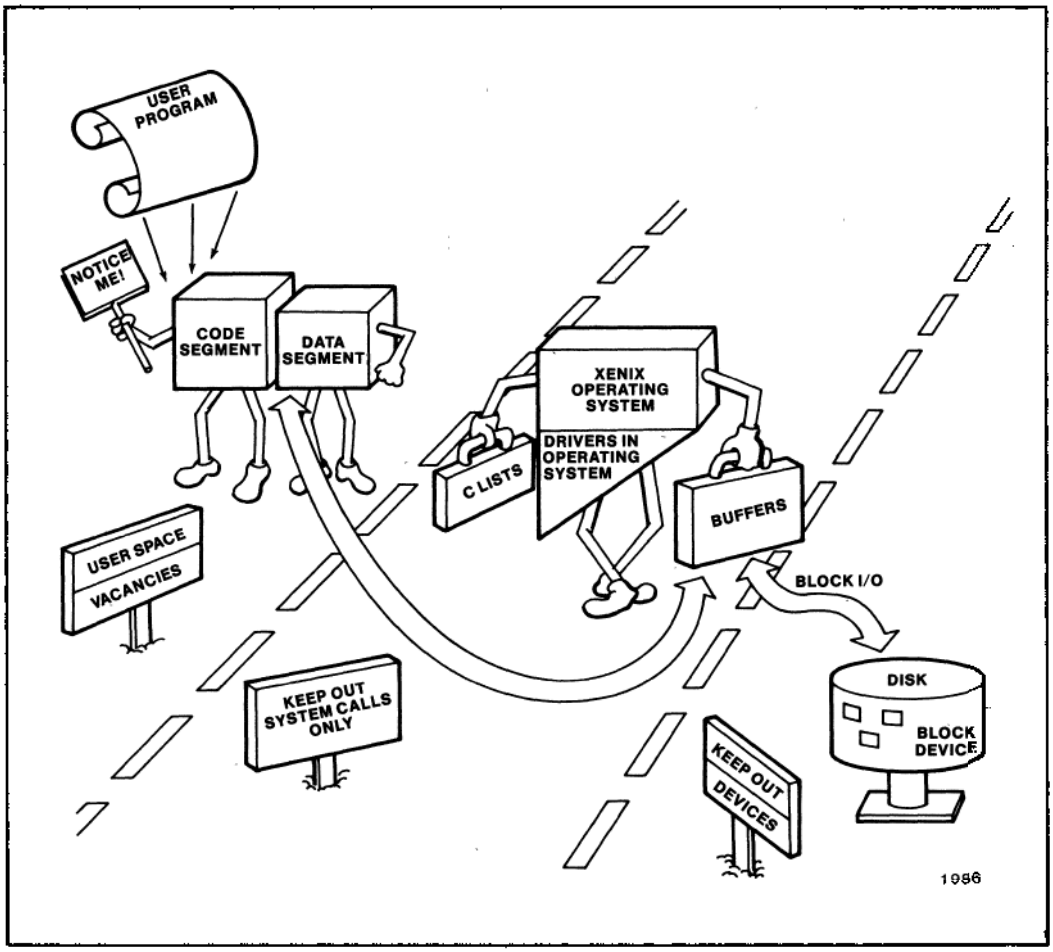


Figure 5b. The System Buffers

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

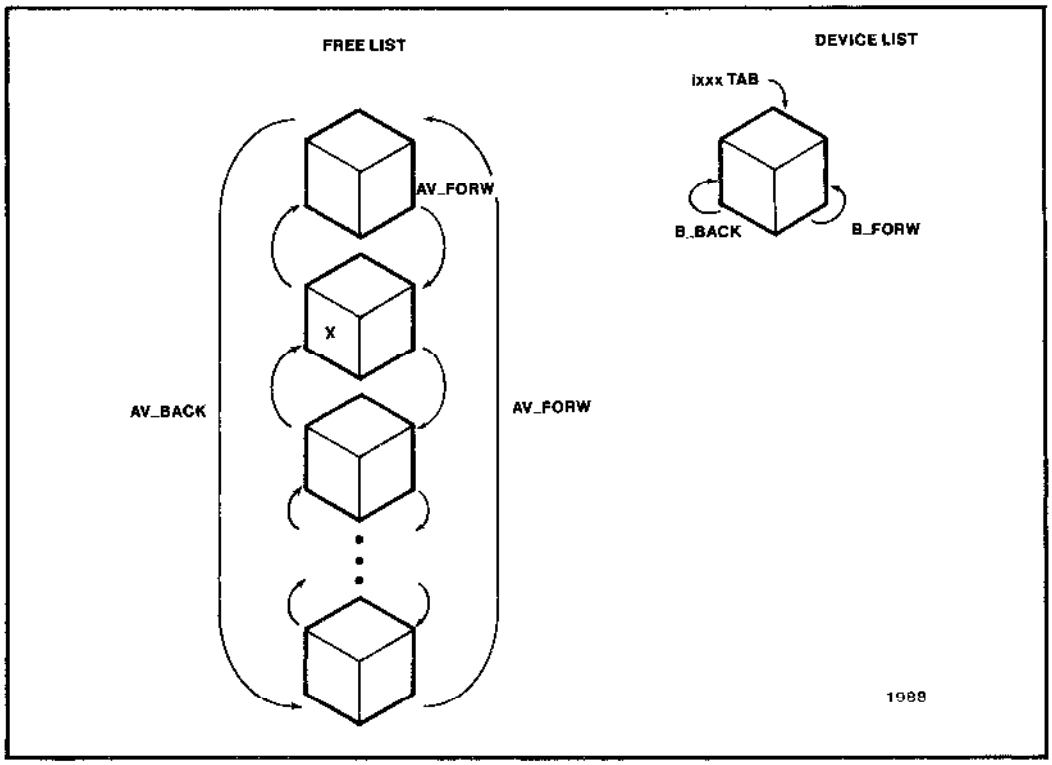


Figure 6a. Start State

xxx is the device identification number e.g. 215, 544. **xxxtab** is the pointer to the system buffers for that device i.e xxxtab is the static buffer header address for the queue of device request. Remember that at start state, these queues are empty.

When a user process requests a write (as a first request) a buffer header from the freelist is removed and placed on the device list. The av-pointers of the freelist are unused and b_pointers are now used in the device list. The av_pointers are re-declared to be **b_actf** and **b_actl** respectively. These links are used to form a new list using the same buffer headers that are in the device queue. Think of two lists i.e. the device list (with items plucked from the freelist) using all four pointers ,thus being members of two lists superimposed (see figure 6b). But why two lists superimposed? Well, at first there were two lists — the freelist and the device specific list (of course, each device has one device specific list but for this discussion let it suffice to have only one device).

Upon a request, the kernel takes a buffer header from the freelist and places it on the device queue. The kernel also places details of the write request into the buffer header fields. As the kernel manages these buffers, the address of this buffer header is given to the device driver. The driver calls a routine **disksort()** which takes the buffer header and orders it in the device specific queue using not the b_pointers but the unused av_pointers. Note that the buffer header is already in the device queue with the **b_back/b_forw** pointers active. **Disksort()** orders these requests into cylinder order on insert using the **av_forw** (b_actf) and **av_back** (b_actl) pointers forming a new list of optimally ordered requests (see figure 7a).

There are three lists formed here:

1) the device specific list hanging off the device driver (cdevsw ==> xxtab)
2) the active list that uses the same elements in the device list but uses the av_pointers ordered by disksort()
3) the freelist of buffer headers using the av_pointers before redeclaration.

Thus, the write request is then ordered into the active list for the device.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

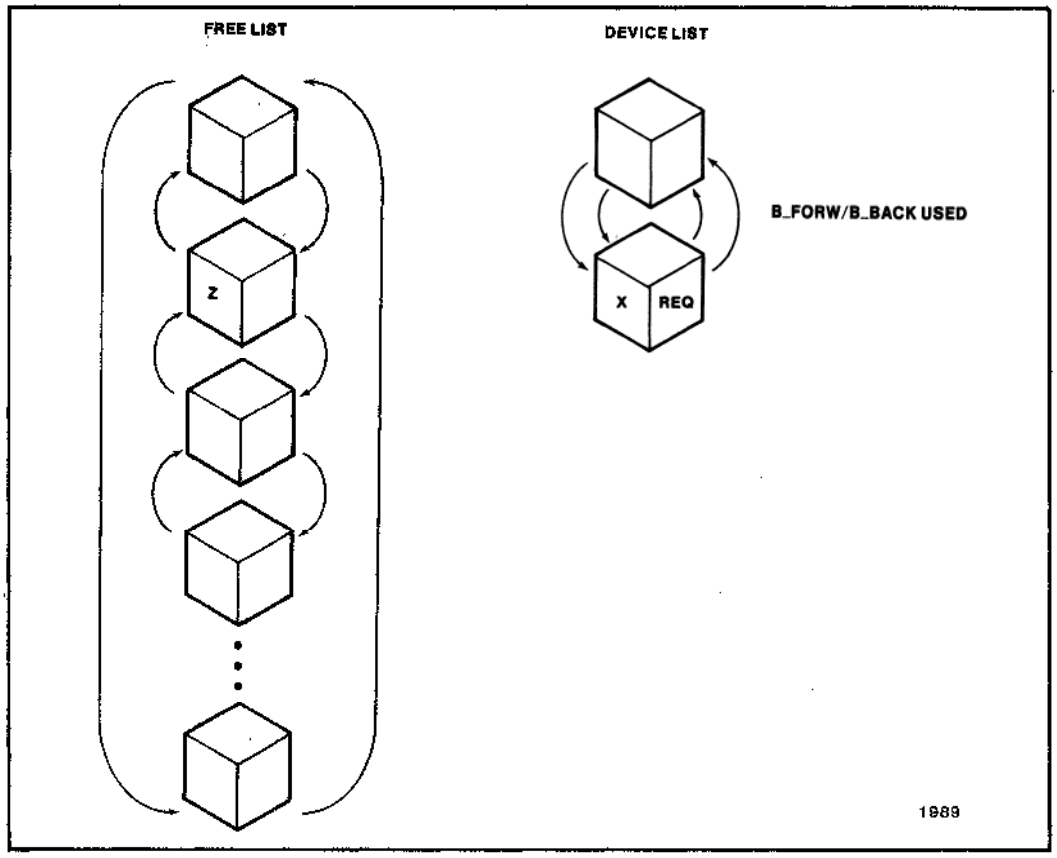


Figure 6b. Buffer placed in device list by kernel

The task-time portion of the driver has been running all this time and this task portion returns after it has made the request. The user process sleeps on the event that the device will complete the requested transaction. In other words, the task sleeps on the buffer-header address. The **sleep()** instruction permits a context switch within the operating system. The operating system can then schedule other tasks for CPU attention.

The device, on completion of the requested write will interrupt the CPU. The interrupt, through the XENIX interrupt scheme, will invoke the respective interrupt service routine which is in the interrupt-time portion of the driver. This is done asynchronously.

This interrupt service routine will awaken all processes associated with the event using **iodone()** (see section 7.1.2). Note that for write, the task-time portion was responsible for the transfer of data from the user space to the buffers while the interrupt service routine is invoked when the data in buffers are written to disk. The relevant process, on awakening is rescheduled by the scheduler to run as soon as possible. The interrupt service routine then checks to see if there are other pending requests on the device specific queue. If not empty, it instigates the next transaction from the next buffer on the device queues (by following the av_pointers now declared **b_actf/b_actl**).

When interrupt routines are alive and running, mutual exclusion(mutex) is ensured by raising the priority level of the running task in the CPU and on completion, lowering the priority level. Details of this technique is found in Interrupt Mapping (Appendix D). This technique locks out interrupts of an equal and higher level than itself for a short period when shared data structures are manipulated. The interrupt service routine may be looking for the next request on the device queue, when the task-portion

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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