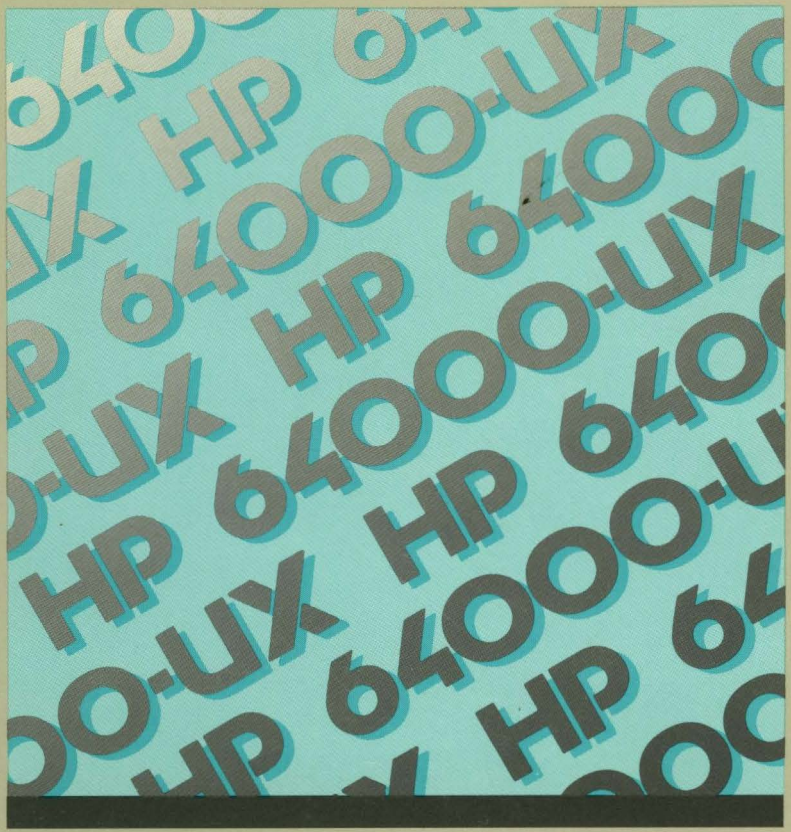
HEWLETT-PACKARD



ADVANCED INTEGRATION ENVIRONMENT

HP 64410 Emulation:

68020 Emulator Operating Manual

64410-90903
E0688

DesignCenter

68020 Emulation: Operating Manual

Edition 1

64410-90903
E0688
Printed in U.S.A. 06/88

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Logic Systems Division
8245 North Union Boulevard
Colorado Springs, CO 80920, U.S.A.

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when as update is incorporated.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

Edition 1 06/88 64410-90903 E0688

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware

designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Radio Frequency Interference

What is Radio Frequency Interference?

All types of electronic equipment are potential sources of unintentional electromagnetic radiation which may cause interference with licensed communication services. Products which utilize digital waveforms such as any computing device are particularly characteristic of this phenomena and use of these products may require that special care be taken to ensure that Electromagnetic Interference (EMI) is controlled. Various government agencies regulate the levels of unintentional spurious radiation which may be generated by electronic equipment. The operator of this product should be familiar with the specific regulatory requirement in effect in his locality.

The HP 64120A Instrumentation Cardcage (ICC) and the HP 64000 Logic Development Station card cage (LDSCC) have been designed and tested to the requirements of the U.S.A. Federal Communications Commission Part 15, Subpart J (Class A), the Federal Republic of Germany FTZ 1046/1984, and the International Electrotechnical Commissions' International Special Committee on Radio Interference (C.I.S.P.R) Publication 22. In addition, the ICC and LDSCC are registered with the Japanese Voluntary Control Council (VCCI). All of these specifications and the laws of many other countries require that if emissions from these products cause harmful interference with licensed radio communications, that the operator of the interference source may be required to cease operation of the product and correct the situation.

Reducing the Risk Of EMI

a. Ensure that the top cover of the ICC or LDSCC is properly installed and that all screws are tight (do not over tighten).

b. When using a feature set which includes cables that egress from the chassis slot of the ICC or LSDCC, ensure that the knurled nuts and ferrels, or brackets that ground the cable shields are clean and tight (Do not overtighten). The EEPROM Programmer cable has an exposed shield that must make contact with the cable clamp.

c. During times of infrequent use, disconnect the EEPROM Programmer and cables from the card cage and the target system.

d. Use only shielded coaxial cables on the four external BNC connectors on the rear of the ICC.

e. Use only the shielded IMB cable supplied with the ICC or LSDCC for connection to additional ICC's or LSDCC's.

f. Use only shielded cables on the IEEE 488 interface connector to the host computer.

Reducing Interference

In the unlikely event that emissions from the ICC or LDSCC result in electromagnetic interference with other equipment, you may use the following measures to reduce or eliminate the interference.

a. If possible, increase the distance between the ICC or LDSCC and the susceptible equipment.

b. Rearrange the orientation of the chassis and cables of the ICC system or LDSCC.

c. Plug the ICC or LDSCC into a separate power outlet from the one used by the susceptible equipment (the two outlets should be on different electrical circuits).

d. Plug the ICC or LDSCC into a separate isolation transformer or power line filter.

You may need to contact your local Hewlett-Packard sales office for additional suggestions. Also, the U.S.A. Federal Communications Commission has prepared a booklet entitled *How to Identify and Resolve Radio - TV Interference Problems* which may be helpful to you. This booklet (stock #004-000-00345-4) may be purchased from the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 U.S.A.

Manufacturer's Declarations

U.S.A. Federal Communications Commission

Warning - This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a class A computing device pursuant to subpart J of Part 15 of the FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential

area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Federal Republic of Germany

Hiermit wird bescheinigt, daß dieses Gerätes in Übereinstimmung mit den Bestimmungen der FTZ 1046/1984 funkentstört ist. Der Deutschen Bundespost wurde das Inverkehrbringen dieses Gerätes angezeigt und die Berechtingung zur Überprüfung der Serie auf Einhaltung der Bestimmungen eingeräumt.

Japan

この装置は，第一種情報装置（商工業地域において使用されるべき情報装置）で商工業地域での電波障害防止を目的とした情報処理装置等電波障害自主規制協議会（ＶＣＣＩ）基準に適合しております。

従って，住宅地域またはその隣接した地域で使用すると，ラジオ，テレビジョン受信機等に受信障害を与えることがあります。

取扱説明書に従って正しい取り扱いをして下さい。

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with

the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

Warning 

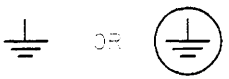
Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

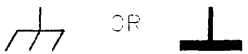
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.



Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.

Alternating current (power line).

Direct current (power line).

Alternating or direct current (power line).

Note 

The Note sign denotes important information. It calls your attention to a procedure, practice, condition, or similar situation which is essential to highlight.

Caution 

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning 

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.

Notice

Caution 

CONDUCTIVE FOAM OR PLASTIC OVER EMULATOR PINS MAY CAUSE ERRATIC OPERATION.

The emulator and preprocessor user assembly pins are covered at the time of shipment with either a conductive foam wafer or a conductive plastic pin protector. This is done for two reasons:

1) to protect the user interface circuitry within the emulator or preprocessor from electro-static discharge (ESD),

2) to protect the delicate gold plated pins of the probe assembly from damage due to impact.

Both the foam and plastic protection devices are conductive. This may cause erratic performance of the emulation or analysis system during operation, and also during option__test performance verification. Therefore, it is recommended that the foam or plastic device be removed before using the emulation or analysis system or before running option__test performance verification.

When not using the emulator or preprocessor, the foam or plastic assembly should be replaced to retain protection for the probe pins and protection from ESD.

Notes

USING THIS MANUAL

Organization

Chapter 1 **Introducing The 68020 Emulator** contains a brief description of the 68020 emulator.

Chapter 2 **Installing Emulation Hardware** contains information on installing your 68020 emulation system hardware into the instrumentation cardcage and making a measurement system. This chapter also contains information on connecting the emulator to your target system.

Chapter 3 **Getting Started** steps you through the emulation process from creating an example program to performing measurements on the execution of that program in emulation.

The **Getting Started** chapter discusses preparing your program modules and the files that are generated by assembling, compiling, and linking programs. See the appropriate cross assembler/linker and compiler manuals for more detailed information on preparing program modules for emulation.

Chapter 4 **Configuring Your Emulator** shows how to access the emulation configuration questions, describes the options available when con-

figuring the emulator, and shows how to load configuration command files from a previous emulation session.

Chapter 5 **Using The Emulator** provides guidelines for using the emulator with a target system and provides information you need to know about how the emulator interacts with your target system.

Chapter 6 **The Emulation Monitor Program** provides a detailed description of the emulation monitor program and how to modify it for your system requirements.

Chapter 7 **Using Custom Coprocessors** describes how to make a custom coprocessor register format file and how to modify the emulation monitor so that your emulation system can display and modify coprocessor registers.

Chapter 8 **Using Simulated I/O And Simulated Interrupts** describes how to set up your emulator to use host I/O resources to simulated target system I/O and how to use the simulated interrupt features of the emulator.

Chapter 9 **How The Emulator Works** provides a detailed description of how many of the emulator features work. Understanding how the emulator works helps you use the emulator more effectively and helps you resolve problems you may encounter.

Appendix A **Emulation Error Messages** contains descriptions of the most serious error messages you may encounter and information on how to correct the errors.

Appendix B provide listings of the demonstration programs used in this manual as a reference for you when working through the the examples.

Appendix C **Timing Comparisons** lists timing comparisons between 68020 processors and the HP 64410 Emulator.

Understanding The Examples

This manual assumes that you are using the User-Friendly Interface Software (HP 64808S) which is activated by executing the HP 64000-UX **pmon** command. This means that the manual will show you how to enter HP 64000-UX system commands (edit, compile, assemble, link, msinit, msconfig, etc.) by telling you to press various softkeys.

If you are not using "pmon", you will find the USER INTERFACE/HP-UX CROSS REFERENCE appendix of the *68020 Emulation Reference Manual* especially useful. The cross reference table shows you how the "pmon" softkeys translate into commands that can be entered from the HP-UX prompt.

The examples provided throughout this manual use the following structure:

PRESS **edit** module.S

PRESS or press means you should enter a command by selecting the softkeys and/or typing in any file names or other variables which are not provided in the softkey selections.

edit softkeys will appear in bold type. Usually you will not be prompted to use the **---ETC---** softkey to search for the appropriate softkey template. Three softkey

templates are available at the HP 64000 system monitor level.

module.S — this is the name of a file which you must type in. Softkeys are not provided for this type of selection since it is variable. However, a softkey prompt such as **<FILE>** will appear as a softkey selection.

For most commands, you must press the **Return** key before the command is actually executed.

Contents

Chapter 1 Introducing The 68020 Emulator

Chapter 2 Installing Your Emulator

Chapter 3 Getting Started

Chapter 4 Answering Emulation Configuration Questions

Chapter 5 Using The Emulator

Chapter 6 The Emulation Monitor Program

Chapter 7 Using Custom Coprocessors

Chapter 8 Using Simulated I/O And Simulated Interrrupts

Chapter 9 How The Emulator Works

Appendix A Emulation Error Messages

Appendix B Source Files For Getting Started Examples

Appendix C Timing Comparisons

Illustrations

Notes

1

Introducing The 68020 Emulator

Overview

This chapter provides the following information:

- Safety considerations for your emulator
- A general description of your emulator
- What information is given in this manual

Safety Considerations

The HP 64000-UX Microprocessor Development Environment, along with the HP 64410SC/SD Emulation Subsystems, is a Class 1 instrument (provided with a protective earth terminal) and meets safety standard IEC 348, "Safety Requirements for Electronic Measuring Apparatus". This Class I instrument meets Hewlett-Packard Safety Class I and has been shipped in a safe condition. Review both the instrument and the manual for safety markings and instructions before operation. Read and become familiar with the "Safety Summary", which follows the Certification/Warranty page of this manual, in addition to the items listed in chapter 2.

Emulator Description

The HP 64410SC/SD Real-Time Emulator for 68020 microprocessors is a powerful tool for both software and hardware designers. Using the HP 64410SC/SD Emulator's emulation memory (up to 512k bytes), software debugging can be done without functional target system memory. Measurements can be made using the emulator's internal 16.7 MHz clock or an external 25 MHz clock with no wait states.

Symbolic debugging lets you debug programs using the same symbols that you defined in your source code. You can control program flow using software breakpoints, single-stepping by op-code, and run-from and run-until commands.

The 68020 emulator has an internal 20 MHz MC68881 Floating Point Coprocessor. You can use this internal coprocessor or an external coprocessor in your target system. The MC68881 instructions can be disassembled in trace displays. You can also display and modify the floating point coprocessor registers.

Dual-ported memory allows you to display or modify emulation memory without halting the processor.

Flexible memory mapping lets you define address ranges referencing emulation memory or target system memory in 256-byte blocks. Blocks can be defined as emulation, target, guarded access, RAM, or ROM over the entire 4 Gbyte address range of the 68020. Since the 68020 supports devices with different memory widths on the data bus, each emulation memory block can be defined as 8, 16, or 32 bits wide.

The HP 64410SC/SD emulator supports the use of 68020 function codes. Emulation memory can be mapped to any of the functional address spaces (CPU, supervisor or user, program or data, or undefined). Function codes can be used as an additional specification when referencing memory.

The integrated emulation bus analyzer provides real-time analysis of all bus-cycle activity. You can define break conditions based on address and data bus cycle activity. In addition to hardware break, software breakpoints can be used for execution breakpoints. You can select any one of the eight 68020 software breakpoint instructions to be used by the emulator.

Analysis functions include trigger, storage, count, and context directives. The analyzer can capture up to 2047 events, including all address, data, and status lines.

Commands for the HP 64410SC/SD emulator and HP 64404A and HP 64405A integrated analyzers have been integrated into one softkey package, making it easy to make both emulation and analysis measurements.

The HP 64410SC/SD emulator can be used for both out-of-circuit emulation and in-circuit emulation. The emulation can be used in multiple emulation systems using other HP 64000-UX Microprocessor Development Environment emulators.

Manual Coverage

This manual provides detailed information on operating the HP 64410SC/SD emulator for the 68020 processor. The information in this manual gives 68020 processor specific information. The *68020 Emulation Reference Manual* provides additional information about using 32-bit emulation, including detailed syntactical descriptions of the emulation commands. Detailed operating information for the HP 64404 and HP 64405 integrated analyzers is given in the *Analysis Reference Manual for 32-Bit Microprocessors* and the *68020 Analysis Specifics* manual.

Notes

2

Installing Your Emulator

Overview

This chapter:

- Reviews the safety considerations for installation.
- Provides preinstallation inspection instructions.
- Shows you how to configure boards in the HP 64120A Instrumentation Cardcage.
- Shows you how to install the emulation system hardware.
- Shows you how to connect the emulation probe cable to your target system.
- Shows you how to turn on the HP 64120A Instrumentation Cardcage.

Introduction

If you are installing your HP 64000-UX components as a new installation, refer to the HP 64000-UX Installation and Configuration Manual for instructions concerning the installation of the HP 64120A Instrumentation Cardcage. Also, refer to the preinstallation instructions given in this section. After you have done these, install the emulation system as instructed later in this section.

Figure 2-1 identifies some key features of the HP 64120A Instrumentation Cardcage. The identifying labels used in this figure are used throughout this manual. Note the location of the power switch. For more information on the hardware configuration, refer to the *Installation and Configuration Manual.*

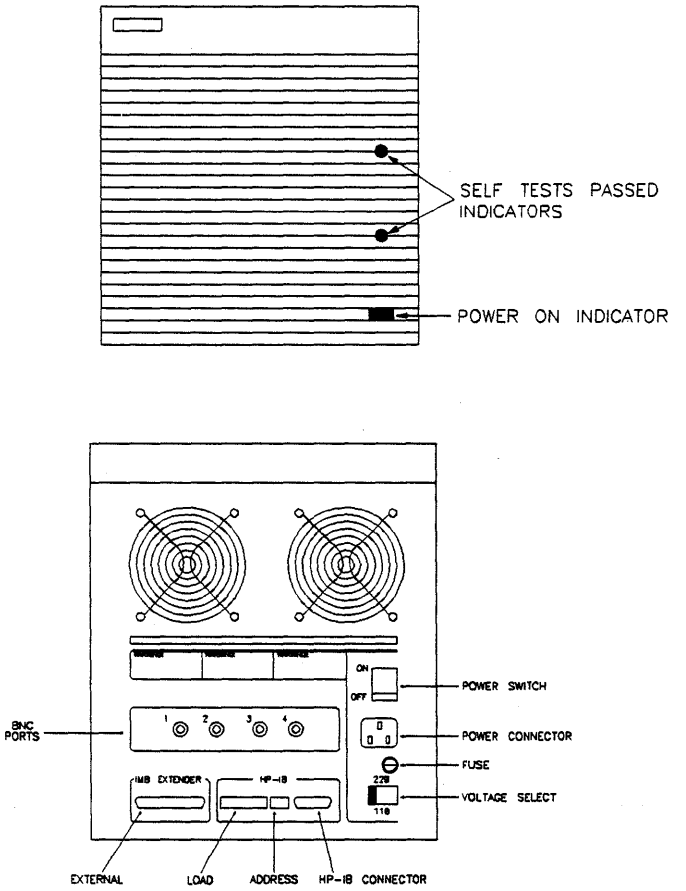


Figure 2-1. HP 64120A Instrumentation Cardcage Features

Safety Considerations

The HP 64000-UX Microprocessor Development Environment along with the HP 64410SC/SD Emulation System is a Class 1 instrument (provided with a protective earth terminal) and meets safety standard IEC 348, "Safety Requirements for Electronic Measuring Apparatus". This Class I instrument also meets Hewlett-Packard Safety Class I requirements and has been shipped in a safe condition.

The user should review both the instrument and manual for safety markings and instructions before operation. Read and become familiar with the "Safety Summary", printed following the Certification/Warranty page of this manual, and the additional items listed below.

Warning 

SHOCK HAZARD! DO NOT ATTEMPT TO DISRUPT PROTECTIVE GROUND! Any interruption of the power cord protective conductor (third prong of power cord plug) inside or outside the HP 64120A Instrumentation Cardcage or disconnection of the protective earth terminal in the power source (wall outlet) is likely to make the HP 64000-UX Microprocessor Development Environment DANGEROUS! Intentional interruption of the power cord protective conductor is prohibited.

Warning 

SHOCK HAZARD! ONLY QUALIFIED PERSONNEL SHOULD SERVICE. Any adjustment, maintenance, or repair of the opened instrument must ONLY be carried out by QUALIFIED PERSONNEL aware of the HAZARDS involved.

Warning 

SHOCK HAZARD! DO NOT USE IF SAFETY FEATURES HAVE BEEN IMPAIRED. If the safety features of the instrument have been damaged or defeated, the instrument shall not be used until repairs are made which restore the safety features. The safety features of the instrument could be disabled in the following instances:

1. The instrument shows visible damage.

2. The instrument fails to perform correct measurements.

3. The instrument has been shipped or stored under unfavorable environmental conditions. Refer to the Service Supplement portion of this manual for information on the environmental specifications of storage and shipment.

Preinstallation Inspection

Unpack all of the emulation system circuit boards, cables, pod, and related equipment. Carefully inspect the equipment for damage that may have occurred during shipping. If any damage is found, please contact your nearest Hewlett-Packard Sales/Service Office as soon as possible.

Verify that all of the items that you ordered have been shipped. If any equipment is missing, please contact your nearest Hewlett-Packard Sales/Service Office as soon as possible.

Installing Your Emulation System Hardware

This section tells you how to install your emulation hardware into the HP 64120A Instrumentation Cardcage.

Warning 

SHOCK HAZARD! INSTALLATION SHOULD ONLY BE PERFORMED BY QUALIFIED PERSONNEL. Any installation, servicing, adjustment, maintenance, or repair of this product must be performed only by qualified personnel. Make sure power is off prior to performing any of the installation instructions given below.

Installation Instructions

Proceed as follows to install the Emulation System and related equipment:

Warning 

SHOCK HAZARD! HAVE YOU READ THE SAFETY SUMMARY? Read the safety summary at the front of this manual before installation or removal of the Emulation Subsystem.

Caution 

DAMAGE TO CARDS AND CAGE! Power to the HP 64120A Instrumentation Cardcage must be removed before installation or removal of option cards (emulation, etc.) to avoid damage to the option cards and the development environment.

Turn Off Power

Turn OFF power to the HP 64120A Instrumentation Cardcage (see figure 2-1 for the location of the power switch on the HP 64120A Instrumentation Cardcage).

Remove The Card Cage Cover

The HP 64120A Instrumentation Cardcage access cover is held in place by four screws on the top of the instrumentation cardcage as seen in figure 2-2. Loosen the four screws, and remove the access cover.

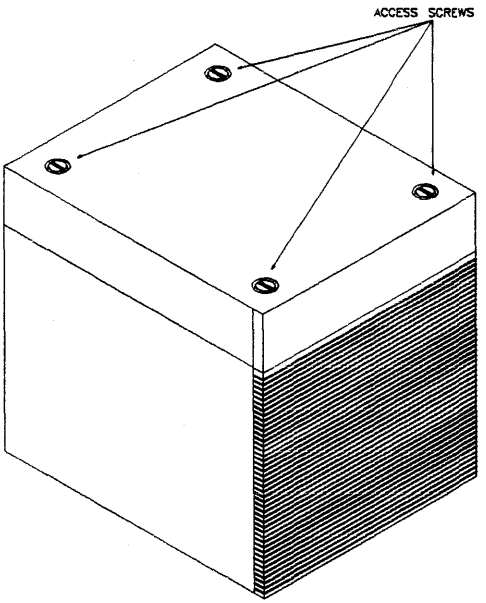


Figure 2-2. Removing the Cardcage Access Cover

Connect The Emulator Pod Cables To The Emulator Boards

There are six cables from the emulation pod that must be connected to various cards in the card cage. Connect these cables as follows:

1. Connect the two 44-conductor cables from the pod to the Emulator Control Board (HP Part Number 64410-66506). There are no color dots to follow because it does not matter which of the 44-conductor cables are connected to each of the 44-pin connectors.

2. Connect the 50-conductor cable from the pod to the Emulator Control Board (HP Part Number 64410-66506).

3. Connect the three 64-conductor cables from the pod to the Analysis Bus Generator board (HP Part Number 64411-66502) following the yellow, red, and brown color dots for proper connections.

The pod cables connected to the ABG board (64411A) are protected by a plastic cover. After connecting the three 64 position cables to the ABG board, secure the plastic cable cover to the ABG board by connecting four screws as shown in figure 2-3. Use a Torx® TX 6 screwdriver.

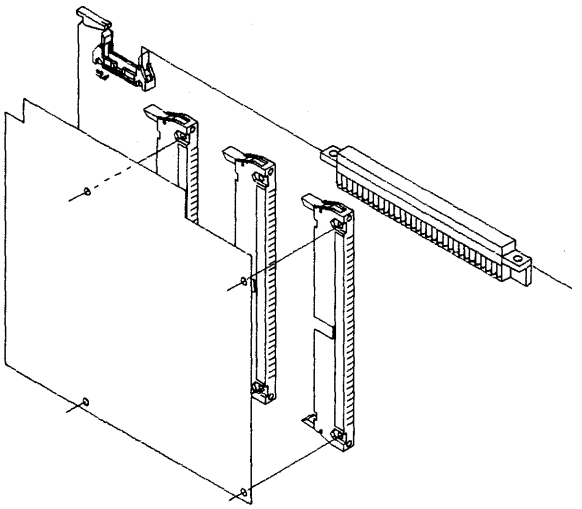


Figure 2-3. ABG Protective Plastic Cable Cover

Install Boards Into The Card Cage

Installation of the circuit boards is accomplished by sliding each circuit board into the circuit board guide slots. As you face the front of the HP 64120A Instrumentation Cardcage, the component side of the boards should face the right side of the instrumentation cardcage. Align the connector at the bottom of the board with the motherboard connector at the bottom of the card cage, then apply a downward pressure until the board is seated in the motherboard connector. Be sure the ejector handles are in their full horizontal position when the board has reached its full downward travel.

Caution

POSSIBLE CABLE DAMAGE! Be careful to avoid scraping the cables or individual wires with the backs of the printed circuit boards. This will strip insulation from the cables and cause short circuits.

Four adjacent card cage slots are required for the circuit boards. Install the boards as follows:

1. Install the boards in the card cage in the order shown in figure 2-4.

2. Install the ABG Interconnect board across the three analysis boards as shown by the "xxxxxx" in figure 2-4.

3. Install the power bus cable between the analysis bus generator and emulator control board as shown by the "oo" in figure 2-4. This bus is not essential, but will improve reliability of the emulator/analyzer system.

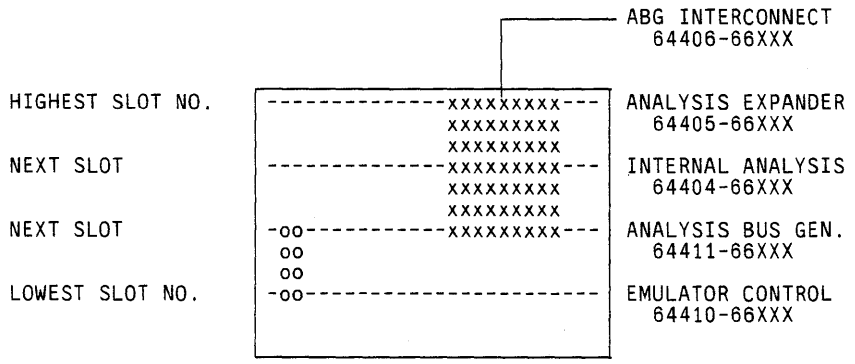


Figure 2-4. Board Installation Into Cardcage

Secure The Pod Cables

Each pod cable has a metal ferrule for strain relief. Snap the ferrule into one of the cable clamps on the instrumentation cardcage. If your instrumentation cardcage does not have cable clamps, you can order them from Hewlett-Packard Co.

Reinstall Card Cage Access Cover

Reinstall the card cage access cover and secure in place with the hold-down screws.

Installing The Emulation Probe Into The Target System

Caution 

PROTECT AGAINST STATIC DISCHARGE! The emulation probe contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the cable from the emulation probe to avoid damaging the internal components of the probe by static electricity.

Caution

POSSIBLE DAMAGE TO EMULATION POD! Do not install the emulation probe into the processor socket with power applied to the target system. The pod may be damaged if power is not removed before installation.
When installing the emulation probe, be sure the probe is inserted into the processor socket so that pin 1 of the emulation probe aligns with pin 1 end of the processor socket. Damage to the emulation equipment may result if the probe is incorrectly installed.

Caution

PROTECT YOUR CMOS TARGET SYSTEM COMPONENTS! If your system includes any CMOS components--turn on the target system first, then turn on the HP 64120A Instrumentation Cardcage; likewise, turn off the development environment first, then the target system.

The emulation probe is provided with a pin protector that prevents damage to the probe when not in use (see figure 2-4). **DO NOT** use the probe without a pin protector installed. If the emulation probe is being installed on a densely loaded circuit board, there may not be enough room to accommodate the plastic shoulders of the probe. If this occurs, another pin protector may be stacked onto the existing pin protector.

To install the microprocessor connector in a target system with a Pin Grid Array (PGA) socket (see figure 2-5), proceed as follows:

1. Remove the 68020 processor from the target system processor PGA socket.

2. Store the 68020 processor in a protected environment. Note the location of pin 1 on both the microprocessor connector and the target system socket.

3. Install the probe cable connector into the target system processor socket.

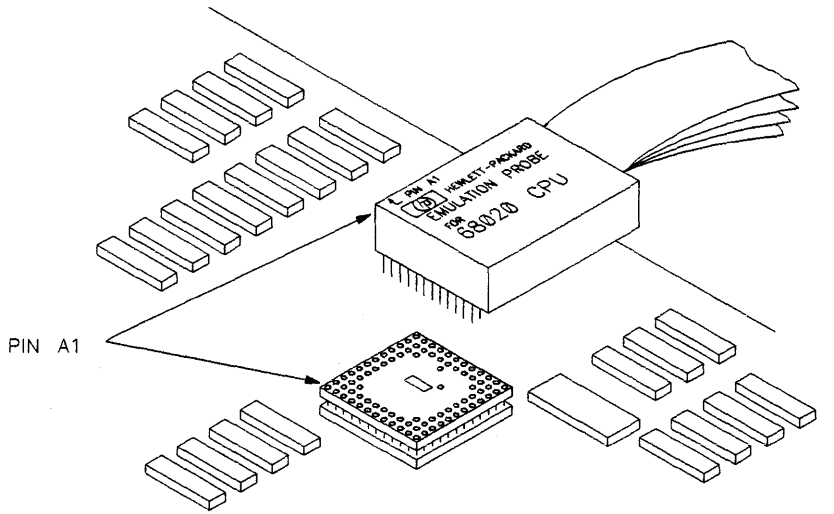


Figure 2-5. Installing Emulator Probe Into PGA Socket

Caution 

PROTECT PGA PINS FROM DAMAGE! To avoid damaging the PGA (Pin Grid Array) probe connector pins, use an insertion/extraction tool (such as Augat P/N TX 8136-13) for removing the PGA probe connector.

Install Software

Refer to the Installation Notice that you received with your HP 64000-UX media for complete software installation instructions.

Installing 68020 Emulation Software Updates

After installing a new copy of the 68020 Emulation Software on a system, cycle the power off and then back on for all HP 64120 cardcages containing 68020 emulators. This updates and initializes all emulation software data structures.

When installing a different revision of the 68020 emulator software, remake all existing configuration files. Configuration file names are suffixed by ".EA" and ".EB". The simplest method is to delete the ".EB" file before loading the configuration file.

Turning On The HP 64120A

The power switch for the HP 64120A Instrumentation Cardcage is identified in figure 2-1.

Caution

PROTECT YOUR CMOS TARGET SYSTEM COMPONENTS! If your system includes any CMOS components--turn on the target system first, then turn on the HP 64120A Instrumentation Cardcage; likewise, turn off the instrumentation cardcage first, then the target system.

Turn the HP 64120A Instrumentation Cardcage power on.

Three green LED's are visible from the front of the HP 64120A Instrumentation Cardcage as seen in figure 2-1. All three should be illuminated to indicate proper operation of the development environment. If all three LED's do not light up, refer to the HP 64120A Instrumentation Cardcage Service Manual for information on correcting any problems.

3

Getting Started

Overview

This chapter describes how to do the following tasks:

- Create a subdirectory in which you can store your 68020 related files.
- Initialize and define a measurement system.
- Assemble, compile, and link the emulation monitor and demonstration programs.
- Access the emulation system from the monitor level softkeys.
- Modify the default emulation configuration and map memory.
- Run an emulation session.

Introduction

This chapter gives an operational overview of the emulation process. The chapter leads you step-by-step through the tasks you must do to prepare your system for emulation and leads you through an emulation session. Emulation features are not explained in depth in this chapter. Its purpose is to familiarize you with the emulation process. Read the entire chapter and go through all exercises in the order presented. This will give you an understanding of the basic operation of the emulator.

Emulation System Used For Examples

The examples given in this chapter (and throughout this manual) were developed with an emulation system including the components listed below.

- HP 64410SD Emulation System
- HP 64870S Cross Assembler/Linker for MC68020
- HP 64903S 68020 C Cross Compiler

Making A Subdirectory For Your 68020 Project

Before you start a new project, make a subdirectory for the project. This enables you to keep your files for each project separate from other files. Follow the rules listed below when you make your subdirectory.

- Give the subdirectory a name consisting of from one to fourteen characters. If more than fourteen characters are used, all characters after the fourteenth character are truncated.
- Any characters may be used in the name. Avoid conflict with special characters used in the HP-UX system software by restricting your subdirectory names to alphanumeric characters and the underscore (_) character.
- Upper and lower case alphabetic characters are significant, i.e., "FILENAME" is a different name than "filename".

Note

The path **/usr/hp64000/bin** must be added to the PATH parameter in your ".profile" file in order to execute HP 64000-UX commands as given in the examples in this manual. Otherwise, you must type the entire path name for HP 64000-UX commands, e.g., **/usr/hp64000/bin/pmon** instead of **pmon**.

Do the following steps to make a subdirectory for your 68020 project:

1. Log in to the system using your login and password.
2. Enter **pmon Return**. This accesses the HP 64000-UX system monitor. The HP 64000-UX system monitor is softkey driven. You should see softkey labels displayed on your screen.
3. Press the **---ETC---** softkey repetitively until the **makedir** softkey appears as an option on the softkey label line.
4. Press the **makedir** softkey and type in the name you wish to use for your directory (the name **em68020** is used throughout this manual). Press the **Return** key on the keyboard.

   makedir em68020 **Return**

You now have a subdirectory named **em68020**.

Whenever you log in to your system to work on the 68020 project, you should change to this directory (using the **chng_dir** softkey). If you do most of your work on the 68020 project, you can modify your ".profile" file to change to this directory whenever you log in. If the permissions are set so that you can alter your own ".profile" file, add the line "cd $HOME/em68020" to your ".profile" file. You will then be in the new subdirectory each time that you log in. If the permissions are set so that you cannot modify your ".profile"

file, see your HP-UX system administrator. The examples in this manual use the **chng_dir** command to change directories.

Initializing And Configuring Your Measurement System

Note

If you have already initialized the instrumentation cardcage and defined your measurement system, skip this section and go to the next section titled "Preparing Your Program Modules".

Refer the *Measurement System manual* for the HP 64000-UX Microprocessor Development Environment for detailed information on initializing and configuring measurement systems. The following procedure gives you a brief overview of the initialization and configuration process.

To initialize your HP 64120A Instrumentation Cardcage and configure your 68020 emulation system, do the following steps:

1. Press **MEAS_SYS**.

Note 

The **MEAS_SYS** softkey is displayed after you enter the HP 64000-UX system monitor by executing the **pmon** command.

You are now in the measurement_system application. The softkeys displayed at this level enable you to initialize and configure your measurement system.

2. Press **msinit Return**.

   If you have only one system in your instrumentation cardcage, the softkey label line will disappear and the message "Working" will appear on the STATUS line. After a few seconds, the message "Hit return to continue" will appear under the STATUS line. Press **Return**. The message will disappear and the softkey labels will return.

   If you have more than one system in your instrumentation cardcage, the softkey label line will disappear and the message "Working" will appear on the STATUS line. After a short time, a list of boards in the card cage may be displayed on the screen. Messages may appear on screen asking you to identify the boards in the different systems. After you have identified any boards requested by the system, the message "Hit return to continue" will appear under the STATUS line. Press **Return**. The message will disappear and the softkey labels will return.

3. Press **msconfig Return**.

   The screen now displays the module(s) available to be assigned (top of the screen) to a measurement system (middle of the screen).

4. Enter **make_sys** emul682k **Return**.

5. Press **add**. If your 68020 emulator is the only system in the instrumentation cardcage, it will be assigned as module 0 as shown at the top of the display. If more than one system is installed in the instrumentation cardcage, the 68020 system module number may be different from 0. Identify the module number of the 68020 emulator shown at the top of the display and type it in from the keyboard. Press **name_it**, type in **em68020** from the keyboard, and press **Return**.

   add 0 **naming_it** em68020 **Return**.

6. Press **end Return**.

   This command causes the system to exit the measurement configuration mode and return to the measurement system level.

7. Press **-GOBACK-** to exit the measurement system level and return to the HP 64000-UX system monitor.

The 68020 Emulation module is now be defined as module **em68020** in the measurement system (shown in the center of the screen).

Preparing Your Program Modules

Program modules must be assembled or compiled, linked, and then mapped to emulation or target memory before the absolute code can be loaded into the emulator. The memory mapping procedure is described briefly in this chapter and is described in detail in chapter 4. The assembly and compile procedures are not described in this manual. Refer to your assembler/linker/librarian and compiler manuals for detailed instructions on these processes.

The following procedures require the HP 64870S Assembler/Linker/Librarian for 68000/10/20 and the HP 64903S 68020 C Cross Compiler. If you have these products on your system, go to the the section titled "Copying the Demonstration Programs to Your Subdirectory".

If you do not have these products on your system, you can still perform the demonstration emulation procedures in this manual. A complete set of files required to perform the emulation examples in this manual are provided on the media with your emulation software. The file set is located in directory **/usr/hp64000/demo/emul32/hp64410**.

You must change to the directory containing the demonstration files in order to run the emulation examples. To change directories, press the **chng_dir** softkey and enter the directory pathname **/usr/hp64000/demo/emul32/hp64410**. The command line should appear as follows:

```
cd /usr/hp64000/demo/emul32/hp64410
```

Press the **Return** key. You should now be in the 68020 demo subdirectory. You can verify this by executing the HP-UX **pwd** (present working directory) command.

Go to the section titled "Preparing The Emulation System".

Note

The README file in the demo directory contains more information on the demonstration files. To read the README file, enter the command:

```
!more /usr/hp64000/demo/emul32/hp64410/README
```

Copying The Demonstration Programs To Your Subdirectory

The demonstration programs used in this manual are provided on the media shipped with your 68020 emulation system in directory /usr/hp64000/demo/emul32/hp64410. The programs are:

simint.c	Simulated interrupt routines for the demonstration program.
towers.c	The demonstration program. This program solves the popular "Towers of Hanoi" brain teaser puzzle. The program demonstrates many features of the emulator, including simulated I/O and simulated interrupts.

Listings of the demonstration programs are included in appendix C of this manual.

Enter the following commands to copy the programs to your subdirectory.

copy /usr/hp64000/demo/emul32/hp64410/simint.c simint.c **Return**
copy /usr/hp64000/demo/emul32/hp64410/towers.c towers.c **Return**

Compiling and Linking the Program Modules

The following sections give examples of how to compile user programs, and link the user programs and emulation monitor into a single executable file. Refer to your compiler manuals for detailed information on these processes.

Compiling The Demonstration Programs.

Enter the following command to compile the demonstration program towers.c:

!cc68020 -hLc towers.c **Return**.

When the message "Hit return to continue" is displayed, Press **Return** and enter the following command to compile the simulated interrupt routine simint.c:

!cc68020 -hLc simint.c **Return**.

When the message "Hit return to continue" is displayed, press **Return**. Your demonstration program files are now compiled.

The **-L** assembler option causes a listing file to be generated.

The **-h** option causes an HP 64000 format assembler symbol file (.A extension) to be generated for debugging purposes. This file is used by the emulator for symbolic debugging.

The **-c** option suppresses automatic linking of the programs (object files are generated).

Enter the following command:

list_dir Return.

Note that four filetype extensions are listed for each compiled file:

c: source file
o: relocatable object file
A: assembler symbols file
O: compiler listing file.

Linking Modules

After you have compiled your source programs, they must be linked, together with the emulation monitor and any required library routines, into an executable module. 68020 emulation environment dependent routines and library routines needed for the

demonstration program are provided with the HP 64903S 68020 C compiler.

The environment dependent routines provide for program setup, dynamic memory allocation, and program input and output in the 68020 emulation environment. These routines are found in directory **/usr/hp64000/env/hp64410**. See the HP 64903S 68020 C Compiler manuals for detailed information on environment dependent routines.

The linker can be used interactively or with a command file to link your programs.

A linker command file that will work with the demonstration program is provided with the HP 64410 demonstration software. The file is named **towers.k**. Copy the file into your subdirectory using the command:

copy /usr/hp64000/demo/emul32/hp64410/towers.k towers.k **Return**

You can view the command file on your screen by entering the command:

!more towers.k **Return**

The program shown in figure 3-1 should be listed on your screen.

Note that the linker command file links a modified version of the emulation monitor from the compiler environment directory. This modified monitor supports the emulator features required by the demonstration programs. See your HP 64903 C compiler manual for detailed information.

```
************************************************************************
* LSD:@(#) .....................................
* @(mktid) .....................................
*
* This is a modified version default linker command file for the
* HP 64903 Advanced 68020 C Cross Compiler to be used with the HP 64410
* 68020 Emulator/Analyzer demonstration software.  It should be used
* along the emulator configuration file config.EA.
*
* NOTE: Revisions 2.00 of the HP64903 Compiler and 1.20 of the HP64870
*       Assembler/Linker or later are required.
************************************************************************
CHIP  68020
SECT  env=$400              * Load address for program/const sections
ORDER env,prog,simint,const,lib,libc,libm
SECT  mon=$20000       * Load address for emulation monitor sections
ORDER mon,mondata
SECT stack=$7FFF8000       * Load address for stack section
SECT envdata=$FFFEA000     * Load address for data sections
ORDER envdata,data,libdata,libcdata,libmdata,heap
************************************************************************
* Set register A5 to the beginning address of the data section + 32k
* so that the A5-relative address mode may be used. If this directive
* is omitted ?A5 has an undefined value.
************************************************************************
INDEX ?A5,data,$8000
LOAD /usr/hp64000/env/hp64410/crt0.o
LOAD /usr/hp64000/lib/68020/libc.a
LOAD /usr/hp64000/lib/68020/lib.a
LOAD /usr/hp64000/env/hp64410/monitor.o
LOAD /usr/hp64000/env/hp64410/env.a
END
```

Figure 3-1. Towers.k Linker Command File

Enter the following command to link your program modules:

!ld68k -Lh -c towers.k -o towers.X simint.o towers.o > towers.MAP **Return**

After a few seconds, the message "Hit return to continue" will appear under the STATUS line. Press **Return**. The message will disappear and the softkey labels will return.

You will use the information contained in the linker listing file towers.MAP when you map emulation memory in the next section.

Preparing The Emulation System

Preparing the emulation system consists of the following steps:

1. Plugging the emulator probe into your target system (for in-circuit emulation).
2. Accessing the emulator through the MEAS_SYS application.
3. Modifying the default emulation configuration to match your system requirements.
4. Loading your user program into emulation or target system memory.

The following procedures use the emulator in out-of-circuit mode (no target system). Target system plug-in issues are discussed in detail in chapter 5 of this manual.

Accessing The Emulation System

Access your emulation system as follows:

1. Press MEAS_SYS.
2. Press emul682k em68020 Return.

You are now in the emulation system application. The emulation softkeys are displayed at the bottom of your screen.

Modifying The Default Emulation Configuration

You accessed the emulator through the use of the default emulation configuration file supplied with your system. You will need to modify this default emulation configuration according to your specific needs. You may need to map memory according to ORG statements in you program or addresses you have specified during the linking process.

Modify the answers to the emulation configuration questions as shown below in order to have the demonstration program run properly. To modify to the emulation configuration questions, enter the command:

modify configuration Return

The first emulation configuration question should be displayed. If not, return to the previous section titled " Accessing the Emulation System" and repeat the steps described there.

Answer the emulation configuration questions as follows.

Emulation Configuration Question	Your Answer
Restrict to real-time runs? no *(Enable the emulation system to break to the emulation monitor)*	Return
Disable breaks into monitor? no *(Enable all emulation functions)*	Return
Reset into the monitor? yes *(A reset command, followed by a run command, will cause the processor to begin executing the emulation monitor)*	Return
Enable emulator use of software breakpoints? yes *(Enable emulator software breakpoints)*	Return
Software BKPT instruction number (0..7)? 7 *(Set the emulation software breakpoint number to 7)*	Return
Enable internal 68881 FPU? no *(Enable emulator use of the internal 68881 FPU in the emulation pod)*	yes Return
FPU coprocessor ID (1..7)? 1 *(Use the default coprocessor ID of 1)*	Return
Name of custom register format file? /usr/hp64000/inst/emul32/0400/0001/custom_spec *(Use the default coprocessor format specification file)*	Return
Modify memory configuration? no *(Modify the emulation memory map)*	yes Return
Break processor on write to ROM? yes *(Break to the emulation monitor if the processor attempts to write to memory mapped as emulation or target ROM)*	Return
Enter the following memory map commands.	
	delete all Return *(Delete all user defined entries)*
	Modify default guarded Return *(Map all unassigned memory locations as guarded, i.e., not accessible to the processor)*
	Modify defined_codes none Return *(Disable emulator use of function code lines)*
	map 0 thru 01efffh emulation rom width32 Return *(Map memory locations for the program and constants sections of the demonstration module)*

Emulation Configuration Question (Continued)	Your Answer
	map 020000h *thru* 022fffh **emulation ram width32 Return** *(Map memory locations for the emulation monitor)*
	map 07fff8000h **thru** 07fffffffh **emulation ram width32 Return** *(Map memory locations for the program stack)*
	map 0fffea000h **thru** 0ffffffffh **emulation ram width32 Return** *(Map memory locations for the data sections of the demonstration module)*
	end Return *(End memory configuration)*
Modify emulator pod configuration? no *(Use the default emulation pod configuration)*	**Return**
Modify simulated I/O configuration? no *(Modify the simulated I/O configuration)*	**yes Return**
Enable polling for simulated I/O? no *(Enable the emulation software to read the simulated I/O control address to determine if the demonstration program has requested any simulated I/O commands)*	**yes Return**
Function code data space ? none *(Use the default value)*	**Return**
Simio control address 1? SIMIO_CA_ONE_systemio_buf *(Specify the control address defined in the demonstration program)*	**Return**
Simio control address 2 .. 6? SIMIO_CA_XXX *(Select the default value for the remaining simio control address questions)*	**Return**
File used for standard input? /dev/simio/keyboard *(Select the host keyboard for simulated I/O input)*	**Return**
File used for standard output? /dev/simio/display *(Select the host display for simulated I/O output)*	**Return**
File used for standard error? /dev/simio/display *(Select the host display as the simulated I/O error output)*	**Return**
Modify simulated interrupt configuration? no *(Modify the default simulated interrupt configuration)*	**yes Return**
Enable polling for simulated interrupts? no *(Enable the emulation software to read the simulated interrupt control address to determine if the demonstration program has requested any simulated interrupt commands)*	**yes Return**

Emulation Configuration Question (Continued)	Your Answer
Function code data space? none *(Select the default value)*	**Return**
Simulated interrupt control address? SIMINT_CA_sim_int_ca *(Specify the control address defined in the demonstration program)*	**Return**
Maximum delay (in milliseconds) for simulated interrupt? 25 *(Specify 3000 milliseconds)*	3000**Return**
Configuration file name? *(Name the configuration file democonfig)*	democonfig **Return**

When the emulator is finished loading the memory mapper, the STATUS line will indicate that the emulation processor is Reset. The emulator is ready to be used.

Note

You now have two configuration files named **democonfig** in your directory. The .EB file extension is a binary file used by the emulator. The .EA file extension is an ASCII file that you can edit using an editor residing on your host system. The emulation configuration file provides you with an easy method to reconfigure your emulator upon entry to the emulation application. Upon reentry to the emulator, enter the command:

load configuration democonfig **Return**

The emulation configuration will be restored to that which you defined in the configuration session you just completed.

Loading Emulation Memory

You are now at the beginning of an emulation session. Before performing emulation, you must load emulation memory with the absolute file created when you linked your program modules. To load emulation memory, enter the following command:

load memory emulation towers **Return**.

Using The Emulator

This section demonstrates the use of some of the basic emulator commands. Work through the examples in the sequence given in this section. Otherwise, the displays you get on your workstation screen may not be the same as those shown in the manual. After you have worked through the examples in this section, you may then execute other commands to gain a better understanding of the emulator's operation. See the *68020 Analysis Specifics manual* and the *Reference Manual for 16- and 32 -Bit Internal Analysis* for detailed information on using the emulator's analysis features.

Note 

The displays you obtain on your system for the examples in the following sections of this chapter may vary from those shown in this manual, depending on the type of terminal or workstation you are using.

Displaying Global Symbols

The **display global_symbols** command displays global (externally defined) symbols in the program modules you have loaded into emulation or target memory. To display global symbols, enter the following command:

display global_symbols Return.

You should see a display similar to the following display on your screen

```
Global symbols in towers
Procedure symbols
Procedure name ____________ Address range _____ Return _ Segment ________ Offset
_startup              000005A0-  00000711    00000710   COMM          00000000
clear_screen          00000970-  000009AB    000009AA   PROG          000001E8
close                 00000828-  00000863    00000862   PROG          000000A0
disable_int           00001822-  0000183B    00001834   PROG          0000001E
enable_int            00001804-  00001821    0000181A   PROG          00000000
exec_cmd              000009F6-  00000AB5    00000AB4   PROG          0000026E
initsimio             000007A0-  000007CD    000007CC   PROG          00000018
kill                  00000AB6-  00000AF9    00000AF8   PROG          0000032E
lseek                 00000B4A-  00000BF1    00000BF0   PROG          000003C2
main                  000010A8-  00001189    00001182   PROG          00000000
open                  000007CE-  00000827    00000826   PROG          00000046
pos_cursor            000009AC-  000009F5    000009F4   PROG          00000224
read                  00000864-  000008F3    000008F2   PROG          000000DC
unlink                00000AFA-  00000B49    00000B48   PROG          00000372
wait_for_io           00000788-  0000079F    0000079E   PROG          00000000

STATUS:  M68020--Reset_____________________________________________...R....
 display  global_symbols
```

You can use the **UP** and **DOWN** cursor keys and the **NEXT** and **PREV** keys to scroll or page through the global symbols listing.

Displaying Local Symbols

You can view local symbols within a file or module using the **display local_symbols_in** command. To view local commands in the demonstration program, enter the following command:

display local_symbols_in towers.c: **Return**.

```
Symbols in towers.c:
Procedure symbols
Procedure name ____________ Address range _____ Return _ Segment _______ Offset
ask_for_number          0000118A-  000013B9     000013B2  PROG          000000E2
init_display            00001688-  00001761     0000175A  PROG          000005E0
main                    000010A8-  00001189     00001182  PROG          00000000
move_disc               00001604-  00001687     00001680  PROG          0000055C
pause                   000013BA-  0000140F     00001408  PROG          00000312
place_disc              0000159A-  00001603     000015FC  PROG          000004F2
remove_disc             00001538-  00001599     00001592  PROG          00000490
show_discs              00001410-  00001537     00001530  PROG          00000368
towers                  00001762-  00001801     000017FA  PROG          000006BA

Static symbols
Symbol name _____________ Address range _____________ Segment _______ Offset
OString1                     000018E4                 COMM          00000084
OString10                    00001A3E                 COMM          000001DE
OString11                    00001A4D                 COMM          000001ED

STATUS:  M68020--Reset_____________________________________________...R....
 display  local_symbols_in towers.c:
```

Note that the **".c"** file extension is used to specify C language files and the **".s"** file extension is used to specify assembly language files.

Displaying Memory

The display memory command enables you to view the contents of either emulation or target memory locations. Enter the command:

display memory main **mnemonic Return**

```
Memory    :mnemonic
     address    data
      10A8  4E560000   LINK.W        A6,#$0000
      10AC  2F0B       MOVE.L        A3,-(A7)
      10AE  2F0A       MOVE.L        A2,-(A7)
      10B0  247CFFFE+  MOVEA.L       #$FFFEA1B4,A2
      10B6  267C0000+  MOVEA.L       #$000013BA,A3
      10BC  42B9FFFE+  CLR.L         $FFFEA1B0
      10C2  60FF0000+  BRA.L         $00001158
      10C8  4E71       NOP
      10CA  7001       MOVEQ         #$00000001,D0
      10CC  2040       MOVEA.L       D0,A0
      10CE  4850       PEA           (A0)
      10D0  4EB90000+  JSR           $00000970
      10D6  588F       ADDQ.L        #4,A7
      10D8  42B9FFFE+  CLR.L         $FFFEA1B8
      10DE  7000       MOVEQ         #$00000000,D0
      10E0  2040       MOVEA.L       D0,A0

STATUS:  M68020--Reset_____________________________________________...R....
 display  memory main  mnemonic
```

The first address listed in the display is 10A8h, the address corresponding to the local symbol **main** in the local symbols display of the towers program. Use the **UP** and **DOWN** cursor keys and the **NEXT** and **PREV** keys to scroll or page through the memory display.

Modifying Memory

You can modify emulation memory locations mapped as either RAM or ROM. The speed of the towers demonstration program is controlled by the variable **loc_delay**. We will set the value of **loc_delay** to 0 so that the program runs at maximum speed. In order to watch the memory display change as the variable is modified, we will display an area in memory repetitively and then modify the memory. Enter the following command:

display memory loc_delay **long repetitively Return**

You should see a display similar to the following on your workstation screen.

```
Memory    :long words :blocked :repetitively
     address    data       :hex                          :ascii
      1860-6F    000001F4 20202020 2020207C 7C202020     ....          ||
      1870-7F    20202020 20202020 2020317C 7C312020                 1||1
      1880-8F    20202020 20202020 2032327C 7C323220                22||22
      1890-9F    20202020 20202020 3333337C 7C333333               333||333
      18A0-AF    20202020 20202034 3434347C 7C343434             4  444||444
      18B0-BF    34202020 20203535 3535357C 7C353535     4     55  555||555
      18C0-CF    35352020 20363636 3636367C 7C363636     55   666 666||666
      18D0-DF    36363620 37373737 3737377C 7C373737     666 7777  777||777
      18E0-EF    37373737 09095075 7A7A6C65 20776974     7777..Pu  zzle wit
      18F0-FF    68202564 20646973 63732063 616E2062     h %d dis  cs can b
      1900-0F    6520736F 6C766564 20696E20 2564206D     e solved   in %d m
      1910-1F    6F766573 2E202020 20200A00 0A0A4578     oves.       ....Ex
      1920-2F    65637574 6520276D 6F646966 79206B65     ecute 'm  odify ke
      1930-3F    79626F61 72645F74 6F5F7369 6D696F27     yboard_t o_simio'
      1940-4F    20746865 6E20656E 74657220 6F6E6520      then en  ter one
      1950-5F    6F662074 68652066 6F6C6C6F 77696E67     of the f  ollowing

STATUS:  M68020--Reset_____________________________________________...R....
 display  memory loc_delay  long  repetitively
```

Enter the command:

modify memory long loc_delay **to** 0 **Return**

Note that the first long word in the display (memory location **loc_delay**) now shows a long word value of 00000000h.

```
Memory    :long words :blocked :repetitively
     address    data       :hex                                    :ascii
      1860-6F    00000000 20202020 2020207C 7C202020      ....                  ||
      1870-7F    20202020 20202020 2020317C 7C312020                        1||1
      1880-8F    20202020 20202020 2032327C 7C323220                       22||22
      1890-9F    20202020 20202020 3333337C 7C333333                      333||333
      18A0-AF    20202020 20202034 3434347C 7C343434                    4 444||444
      18B0-BF    34202020 20203535 3535357C 7C353535      4     55      555||555
      18C0-CF    35352020 20363636 3636367C 7C363636      55   666      666||666
      18D0-DF    36363620 37373737 3737377C 7C373737      666 7777      777||777
      18E0-EF    37373737 09095075 7A7A6C65 20776974      7777..Pu      zzle wit
      18F0-FF    68202564 20646973 63732063 616E2062      h %d dis      cs can b
      1900-0F    6520736F 6C766564 20696E20 2564206D      e solved       in %d m
      1910-1F    6F766573 2E202020 20200A00 0A0A4578      oves.           ....Ex
      1920-2F    65637574 6520276D 6F646966 79206B65      ecute 'm      odify ke
      1930-3F    79626F61 72645F74 6F5F7369 6D696F27      yboard_t      o_simio'
      1940-4F    20746865 6E20656E 74657220 6F6E6520       then en      ter one
      1950-5F    6F662074 68652066 6F6C6C6F 77696E67      of the f      ollowing

STATUS:  M68020--Reset_____________________________________________________...R....
 modify  memory  long loc_delay  to 0
```

Running from the Transfer Address

Now that you have used some of the display and modify features of the emulator, it is time to run the demonstration program and use some of the run time features of the emulation system. Enter the following command:

run from transfer_address Return

The STATUS line displays **"M68020--Running"**. This indicates that the demonstration program is executing.

Displaying Registers

The **display registers** command enables you to look at the contents of the 68020's CPU registers and the contents of the 68881 floating point coprocessor registers. Enter the following command:

display registers cpu Return

The contents of the following 68020 CPU registers are displayed on the screen:

program counter (PC)
source function code register (SFC)
destination function code register (DFC)
data registers (D0--D7)
address registers (A0-A7)
user stack pointer (USP)
vector base register (VBR)
cache address register (CAAR)
master stack pointer (MSP)
interrupt stack pointer (ISP)
status register (STATUS)
cache control register (CACR)

```
M68020 Registers

 NextPC 00000792      SFC 0 MOT RSVD           DFC 0 MOT RSVD
  D0-D7 00000000 00000092 000003FC 00003248 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA037 FFFEA698 FFFEA78C FFFEA034 FFFF21A8 7FFFFF02
    USP 00020D60      VBR 00000000           CAAR 00000000
    MSP 00020D60            <t1 t0 s  m i  x  n  z  v  c>          <f  e>
   *ISP 7FFFFF02   STATUS 2704 0  0  1  0  7  0  0  1  0  0     CACR 0  0  0




STATUS:   M68020--Running_____________________________________________...R....
 display  registers cpu
```

Press the **break** softkey, then press **Return**.

The registers display is updated and the status line now reads **"STATUS: M68020--Running in monitor"**. If a display registers command has been executed in the current emulation session, the registers display is updated whenever a break to the emulation monitor program occurs.

```
M68020 Registers

 NextPC 00000792      SFC 0 MOT RSVD            DFC 0 MOT RSVD
  D0-D7 00000000 00000092 000003FC 00003248 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA037 FFFEA698 FFFEA78C FFFEA034 FFFF21A8 7FFFFF02
    USP 00020D60      VBR 00000000              CAAR 00000000
    MSP 00020D60                <t1 t0 s  m i  x  n  z  v  c>          <f  e>
   *ISP 7FFFFF02   STATUS 2704 0  0  1  0  7  0  0  1  0  0      CACR 0  0  0

 NextPC 00000790      SFC 0 MOT RSVD            DFC 0 MOT RSVD
  D0-D7 00000092 00000092 000003FC 00003248 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA037 FFFEA698 FFFEA78C FFFEA034 FFFF21A8 7FFFFF02
    USP 00020D60      VBR 00000000              CAAR 00000000
    MSP 00020D60                <t1 t0 s  m  i  x  n  z  v  c>          <f  e>
   *ISP 7FFFFF02   STATUS 2704  0  0  1  0  7  0  0  1  0  0      CACR 0  0 0



STATUS:   M68020--Running in monitor_____________________________________...R....
 break
```

Using The Step Function

The step function enables you to step through your program opcode by opcode. Each time the step command is executed, one program instruction is executed. Enter the command:

step from transfer_address Return

The register display is updated each time a step is executed. In the last entry on the display an additional line is displayed. The address of the instruction executed by the step command and the executed instruction are displayed on the first line of the new register display entry. The step feature is a powerful tool for debugging programs because it enables you to watch the register activity for each executed instruction

```
M68020 Registers

   *ISP 7FFFFF02   STATUS 2704  0  0  1 0  7  0  0  1  0  0      CACR 0  0  0

 NextPC 00000790       SFC 0 MOT RSVD              DFC 0 MOT RSVD
  D0-D7 00000092 00000092 000003FC 00003248 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA037 FFFEA698 FFFEA78C FFFEA034 FFFF21A8 7FFFFF02
    USP 00020D60       VBR 00000000              CAAR 00000000
    MSP 00020D60                <t1 t0 s  m  i  x  n  z  v  c>           <f  e>
   *ISP 7FFFFF02   STATUS 2704  0  0  1  0  7  0  0  1  0  0      CACR 0  0  0

     PC 000004E4   Opcode MOVE.L        A7,$7FFFFFFC                    23CF7FFF
 NextPC 000004EA       SFC 0 MOT RSVD              DFC 0 MOT RSVD
  D0-D7 00000092 00000092 000003FC 00003248 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA037 FFFEA698 FFFEA78C FFFEA034 FFFF21A8 7FFFFF02
    USP 00020D60       VBR 00000000              CAAR 00000000
    MSP 00020D60                <t1 t0 s  m  i  x  n  z  v c>           <f  e>
   *ISP 7FFFFF02   STATUS 2700  0  0  1  0  7 0  0  0  0  0      CACR 0  0  0

STATUS:   M68020--Running in monitor_____________________________________...R....
 step  from transfer_address
```

Enter the command:

step Return

Note that the emulator executes the instruction stored in the NextPC memory location. Press Return repetitively. The emulator executes one instruction each time you press Return.

The step instruction enables you to specify a number of steps. This is useful when stepping through program structures such as delay loops. Enter the command:

step 25 Return

Notice that the screen is updated with register information each time a program step is executed. While the step command is being executed, the status line displays the message **"MC68020--Steps left #n"** where n is the number of steps remaining. You can use the **NEXT** and **PREV** keys and the **UP** and **DOWN** keys to look at register information that has scrolled off of the screen.

Tracing Processor Activity

The trace function (with analyzer present) enables you to watch each cycle on the processor bus as it occurs. The following examples illustrate some simple uses of the trace function. For more information on the trace function, refer to the *Analysis Reference Manual for 32-Bit Microprocessors* and the *68020 Analysis Specifics manual.*

Enter the command:

trace TRIGGER_ON a= long_aligned main **Return**

This sets up a trace of all activity of the bus for 2k bus cycles before and 2k bus cycles after the address labeled main occurs. The STATUS line will indicate "Trace in process". Enter the command:

run from main **Return**

After the STATUS line indicates "Trace complete", enter the command:

display trace Return

The trace list is displayed on the screen with the trigger state displayed in the center of the screen. Notice the lines prior to the trigger state. The address field shows that these lines represent emulation monitor execution and stack accesses. User program activity is displayed starting with the trigger state (000010A8h).

```
Trace List
Label:   Address                    Opcode or Status                    time count
Base:      hex                         mnemonic                         relative
-0007    7FFFFFC4   $2708xxxx     supr data word wr (ds32)               0.20us
-0006    000206F4   $36390002     supr prgm long rd (ds32)               0.24us
-0005    00020DCC   $0000xxxx     supr data word wr (ds32)               0.24us
-0004    7FFFFFC4   $2708xxxx     supr data word rd (ds32)               0.16us
-0003    7FFFFFCA   $xxxx2024     supr data word rd (ds32)               0.20us
-0002    7FFFFFC6   $xxxx0000     supr data long rd (ds32)               0.16us
-0001    7FFFFFC8   $10A8xxxx     supr data word rd (ds32)               0.20us
trigger  000010A8   $4E560000     supr prgm long rd (ds32)               0.40us
+0001    000010AC   $2F0B2F0A     supr prgm long rd (ds32)               0.24us
+0002    7FFFFFCC   $00000000     supr data long wr (ds32)               0.32us
+0003    000010B0   $247CFFFE     supr prgm long rd (ds32)               0.16us
+0004    7FFFFFC8   $FFFEAF94     supr data long wr (ds32)               0.32us
+0005    000010B4   $A1B4267C     supr prgm long rd (ds32)               0.16us
+0006    7FFFFFC4   $FFFEA034     supr data long wr (ds32)               0.16us
+0007    000010B8   $000013BA     supr prgm long rd (ds32)               0.20us

STATUS:   M68020--Running                Trace complete_____________...R....
 display  trace
```

The address and data values in the default trace list are displayed as hexadecimal numbers. The emulator can also display values in assembly language mnemonics. Enter the command:

display trace disassemble_from_line_number 0 Return

```
Trace List
Label:   Address                      Opcode or Status                  time count
Base:      hex                           mnemonic                       relative
trigger  000010A8  LINK.W        A6,#$0000                                0.40us
+0001    000010AC  MOVE.L        A3,-(A7)                                 0.24us
        =000010AE  MOVE.L        A2,-(A7)
+0002    7FFFFFCC    $00000000     supr data long wr (ds32)               0.32us
+0003    000010B0  MOVEA.L        #$FFFEA1B4,A2                          0.16us
+0004    7FFFFFC8   $FFFEAF94      supr data long wr (ds32)               0.32us
+0005   =000010B6  MOVEA.L        #$000013BA,A3                          0.16us
+0006    7FFFFFC4    $FFFEA034     supr data long wr (ds32)               0.16us
+0007    000010B8    $000013BA     supr prgm long rd (ds32)               0.20us
+0008    000010BC  CLR.L          $FFFEA1B0                              0.32us
+0009   =000010C2  BRA.L          $00001158                              0.28us
+0010    000010C4    $00000094     supr prgm long rd (ds32)               0.32us
+0011    FFFEA1B0    $00000000     supr data long wr (ds32)               0.24us
+0012    00001158  PEA             (A2)                                   0.16us
        =0000115A  JSR             $0000118A

STATUS:  M68020--Running                  Trace complete_____________...R....
 display  trace disassemble_from_line_number 0
```

Note that in the updated trace display, the trigger line (line 0) is the first line in the trace display. Address 000010A8h corresponds to the **main** label in the demonstration program. The instruction **MOVE.L** on the trigger line is the first instruction in the example program.

You can also display the source program lines corresponding to the traced assembly level code in the trace list. Enter the command:

display trace source on inverse_video on Return

The display is updated with the source code line displayed in inverse video immediately before the related traced assembly level code.

```
Trace List
Label:   Address          Opcode or Status w/ Source Lines          time count
Base:      hex                      mnemonic                        relative
       ##########towers.c - line     1 thru   126 ###############################
       static void towers();
       static int  ask_for_number();

       main()
       {
trigger 000010A8 LINK.W        A6,#$0000                                0.40us
+0001    000010AC MOVE.L       A3,-(A7)                                 0.24us
       =000010AE MOVE.L        A2,-(A7)
 +0002    7FFFFFCC   $00000000     supr data long wr (ds32)               0.32us
+0003    000010B0 MOVEA.L       #$FFFEA1B4,A2                            0.16us
+0004    7FFFFFC8   $FFFEAF94     supr data long wr (ds32)               0.32us
+0005   =000010B6 MOVEA.L       #$000013BA,A3                            0.16us
+0006    7FFFFFC4  $FFFEA034      supr data long wr (ds32)               0.16us
+0007    000010B8   $000013BA     supr prgm long rd (ds32)               0.20us

STATUS:   M68020--Running                Trace complete_____________...R....
 display  trace  source  on inverse_video  on
```

You can use the **UP** and **DOWN** cursor keys or the **NEXT** and **PREV** keys to scroll or page through the entire trace listing. You can copy the trace list to the printer or a file as well.

Using Software Breakpoints

The set sw_breakpoints command lets you set software breakpoints in your program code. This useful feature lets you break execution of your program at the point you select. You can then use the many display and modify commands available in the emulator to examine and debug your code. The emulator replaces the code at the memory location you specify with a 68020 BKPT instruction. You select the appropriate BKPT instruction when answering the emulation configuration questions. Enter the command:

break Return

You are now running in the emulator monitor program. Enter the command:

modify sw_breakpoints set one_shot
towers.c:ask_for_number **Return**

This command causes the emulator to replace the instruction at the address reference by the symbol **ask_for_number** (0000118AH) with a **BKPT 7** instruction. The address specified in the command must be the first address of an opcode. Enter the following command

: **display memory**
towers.c:ask_for_number **mnemonic Return**

The display shows a **BKPT 7** instruction at address 0000118AH.

```
Memory     :mnemonic
     address     data
      118A  484F        BKPT          #7
      118C  000048E7    ORI.B         #$E7,D0
      1190  3C38247C    MOVE.W        $0000247C,D6
      1194  FFFE        rsvd coproc instr type
      1196  A698        Reserved Instruction: $A698
      1198  267C0000+   MOVEA.L       #$00006214,A3
      119E  287C0000+   MOVEA.L       #$00006264,A4
      11A4  24360961+   MOVE.L        ([$0008,A6]),D2
      11AA  4AB9FFFE+   TST.L         $FFFEA1B0
      11B0  66FF0000+   BNE.L         $0000139E
      11B6  7001        MOVEQ         #$00000001,D0
      11B8  2040        MOVEA.L       D0,A0
      11BA  4850        PEA           (A0)
      11BC  4EB90000+   JSR           $00000970
      11C2  588F        ADDQ.L        #4,A7
      11C4  7007        MOVEQ         #$00000007,D0

STATUS:   M68020--Running in monitor      Trace complete_____________...R....
 display  memory towers.c:ask_for_number  mnemonic
```

Enter the command:

run from transfer_address Return

The demonstration program runs from the transfer_address (_main) until the BKPT instruction is executed. The BKPT instruction causes the emulator to break into the emulation monitor and the message **"STATUS: Software breakpoint hit at address = 118A"** is displayed on the status line.

The emulator's ability to let you set software breakpoints provides you with a method of stopping program execution at a specified point in your program. You can then examine register values, display or modify memory locations, and perform other operations before continuing execution of your program.

Enter the command:

display memory towers.c:ask_for_number **Return**

Note that the instruction **LINK.W** is now displayed at address 118AH in the memory listing. After breaking into the emulation monitor, the emulator replaces the BKPT instruction with the original contents of the memory location (LINK.W instruction).

```
Memory     :mnemonic
     address     data
       118A   4E560000    LINK.W        A6,#$0000
       118E   48E73C38    MOVEM.L       rm=$3C38,-(A7)
       1192   247CFFFE+   MOVEA.L       #$FFFEA698,A2
       1198   267C0000+   MOVEA.L       #$00006214,A3
       119E   287C0000+   MOVEA.L       #$00006264,A4
       11A4   24360961+   MOVE.L        ([$0008,A6]),D2
       11AA   4AB9FFFE+   TST.L         $FFFEA1B0
       11B0   66FF0000+   BNE.L         $0000139E
       11B6   7001        MOVEQ         #$00000001,D0
       11B8   2040        MOVEA.L       D0,A0
       11BA   4850        PEA           (A0)
       11BC   4EB90000+   JSR           $00000970
       11C2   588F        ADDQ.L        #4,A7
       11C4   7007        MOVEQ         #$00000007,D0
       11C6   2040        MOVEA.L       D0,A0
       11C8   4850        PEA           (A0)

STATUS:  M68020--Running in monitor      Trace complete_____________...R....
 display  memory towers.c:ask_for_number
```

To continue execution of your program from the point the break occurred, enter the command:

run Return

Notice that the status line now reads **"M68020--Running"**.

Refer to chapter 9 for a description of how the software breakpoint function is implemented in the 68020 emulator. See chapter 2 of the *68020 Emulation Reference Manual* for the software breakpoint command syntax.

Using Simulated I/O

The demonstration program uses simulated I/O for both entering parameters and displaying the solution to the towers of Hanoi problem. To display the simulated I/O screen, enter the command:

display simulated_io Return

Your screen should appear as shown in the following display.

```
Simulated I/O display                    Simulated I/O command:  read
display is open                                  Return code:  00H


Execute 'modify keyboard_to_simio' then enter one of the following:
        Number of discs to use [1-7]
        '0' to exit program
        'C' to run continuously using last number entered

?









STATUS:   M68020--Running                Trace complete_____________...R....
 display simulated_io
```

The keyboard must be assigned to simulated I/O before it can be used to specify the number of discs to be used in the program. Enter the command:

modify keyboard_to_simio Return

The keyboard is now assigned to simulated I/O and is accessible to the demonstration program. Enter the number **7** and press **Return**.

The program then uses simulated I/O to display the solution to the problem on the screen as shown in the following display.

```
Simulated I/O display                    Simulated I/O command:  write
display is open                                 Return code:  00H

                  ||                              ||                  ||
                  ||                              ||                  ||
                  ||                              ||                  ||
              333 || 333
             4444 || 4444
            55555 || 55555
           666666 || 666666
          7777777 || 7777777              22 || 22                 1 || 1
          ---------------                 ---------------          ---------------
               Peg  0                          Peg  1                   Peg  2

                  Solution for Towers with 7 discs.
                  Move #3:  Move disk 1 from peg 2 to peg 1


STATUS:   M68020--Running                  Trace complete_______________.........
```

To return control of the keyboard to the host system, press the **suspend** softkey. The normal emulation softkeys will be restored.

For more information on using simulated I/O, see chapter 4 and the *Simulated I/O Operating Manual* supplied with your HP 64000-UX system.

Ending The Emulation Session

To end the emulation session, enter the command:

end release_system Return

The system will return to the **MEAS_SYS** application level.

This completes your introduction to the 68020 emulation system. You have assembled and compiled program modules, linked your program modules, and used a few of the basic features of the emulation system. For more detailed operational information, refer to the information contained in the other chapters of this manual and the *68020 Emulation Reference Manual.* See the *Analysis Reference Manual for 32_Bit Microprocessors* and the *68020 Analysis Specifics* manuals for detailed information on the analysis features provided in the emulator.

Using Command Files

A command file is a file that lists a series of commands that must be performed to accomplish a particular function. Command files are ideal for setting up, and accessing, the emulation system. Once the file is created, all you need to do is type the file name and press **Return**. The commands in the file will be executed, allowing you to easily enter your emulation session. Refer to the "Creating and Using Command Files" chapter of the *HP 64000-UX User's Guide* for detailed information on command files.

Notes

4

Answering Emulation Configuration Questions

Overview

This chapter:

- Explains each of the emulation configuration questions.
- Describes how to configure the emulator for compatibility with your 68020 target system.
- Describes how to map your 68020 system memory to emulation and target system memory resources.

Introduction

The 68020 emulator is configured from within the emulation application. When you run emulation for the first time, a default configuration file is loaded. You can modify this file to match your particular system needs by answering a series of emulation configuration questions displayed on your workstation display. After modifying the emulation configuration, you can save it to a file which you can then load each time you enter emulation.

Your answers to the emulation configuration questions define how your 68020 emulator is configured, how resources are shared between the emulator and your target system, how the emulator and target system interact, and what operations are enabled in the emulation environment.

The configuration questions enable you to do the following emulation configuration tasks:

- Selecting real time or nonreal-time run mode.
- Enabling breaks to the emulation monitor.
- Selecting whether to reset into the emulation monitor or to use the user reset exception vector.
- Enabling and selecting the software breakpoint instruction.
- Enabling the internal emulation FPU.
- Configuring custom coprocessor functions.
- Configuring memory.
- Configuring the emulator pod.
- Configuring simulated I/O and interrupts.
- Naming your emulation configuration command file.

Running Emulation

The command sequence to run emulation depends on how you configured your emulation system and what you named it. In this chapter, the example names from chapter 3, Getting Started, are used. To run emulation, do the following steps:

1. Press **MEAS_SYS**.
2. Press **emul682k em68020 Return**.

Modifying The Configuration File

To modify the configuration, enter the following command:

modify configuration Return

A series of questions are displayed on your workstation screen. Your answers to these emulation configuration questions specify the configuration of the emulation hardware and software for a specific application. Each question is displayed with a default response. Additional options are shown in this chapter in parentheses. The default response is selected by pressing the **Return** key. Other responses are selected by pressing the appropriate softkey or by typing in an appropriate response, and then pressing **Return**. If you are modifying an emulation configuration file which you previously made, the default responses are those responses stored in that configuration file.

Note 

If you need to return to a question you have already answered, press the **RECALL** softkey. Each time you press **RECALL**, the emulator backs up to the configuration question that was displayed prior to the question currently displayed. You may then make any corrections needed.

Selecting Real-Time/ Nonreal-Time Run Mode

Real-time refers to the continuous execution of your 68020 program without interference from the development environment except as specified by you. All commands which cause momentary breaks to the emulation monitor are disabled. Momentary breaks are breaks asserted by the emulation software which momentarily diverts 68020 execution to the emulation monitor and then resumes execution of your program. In real-time run mode, you can execute any command which does not cause a break to the emulation monitor. Commands requiring target memory or register accesses are disabled when a user program is running. These commands can only be executed while running in the

emulation monitor. An attempt to execute a **run/step from <ADDR>** command while executing the user program in real time causes a break to the emulation monitor.

If the emulator is not restricted to real-time run mode, all selected emulation functions are enabled. Commands requiring access to target memory or registers cause a break to the emulation monitor if a user program is running.

Features which require emulation monitor interaction interfere with real-time operation more than features which require only emulation memory interaction. A major portion of real-time interference can be avoided by disabling the emulation monitor functions. You can select this option later in the configuration questions.

Restrict to real-time runs? no (yes)

no	All selected emulator functions are enabled. The emulation system is enabled to break to the emulation monitor whenever a command requiring breaks to the emulation monitor is executed.
yes	Target memory and register accesses are disabled when a user program is running.

Caution 

POSSIBLE DAMAGE TO CIRCUITRY! When the emulator detects a guarded memory access or other illegal condition, or when you execute a command that causes the emulator to break into the emulation monitor, the emulator stops executing the user program and enters the emulation monitor. If you have circuitry in your target system that can be damaged because the emulator is not executing your code, you should use caution. Restrict the emulator to run in real-time mode only. Do not execute commands that cause breaks to the emulation monitor.

Enabling Emulator Monitor Functions

The next question asks you if you want to disable breaks into the emulation monitor. If you answer no, all emulation commands and features implemented by the emulation monitor are enabled. If you answer yes, configuration questions that refer to functions requiring the emulation monitor will not be asked. They will be set to the following default values:

Reset into the monitor?	no
Enable emulator use of software breakpoints?	no
Break processor on write to ROM?	no

If the emulation monitor is not loaded, all emulation functions that require the monitor for execution will be disabled and their associated softkeys turned off. The functions that require the emulation monitor are:

- automatic reset to monitor
- break
- copy target memory
- copy registers
- display target memory
- display registers
- emulator use of software breakpoints
- load target memory
- modify target memory
- modify registers
- run from/until <ADDR>
- set break_on
- step
- store target memory

Disable breaks into monitor? no (yes)

no	All emulation commands and features implemented with the emulation monitor are enabled.
yes	Configuration questions that refer to functions requiring the emulation monitor are not asked. If no emulation monitor is loaded, all commands and features requiring the emulation monitor are disabled and their associated softkeys are turned off.

Resetting Into The Monitor

Note 

If you answered **yes** to the previous question, the following question will not be displayed on your screen.

The next question lets you select whether the emulation **reset** command causes the processor to be reset into the emulation monitor or to the memory location specified by the user reset exception vector. This question only affects reset commands entered from the workstation keyboard or processor reset on entry to the emulation module. It has no effect on reset signals generated within the user's target system.

Reset into the monitor? yes (no)

yes	The emulation reset command causes the processor to be reset into the emulation monitor. The user-defined reset vector and initial stack pointer are ignored.

no	The emulation reset command causes the processor to fetch the user-defined reset vector and begin execution from that address.

Enabling Emulator Use of Software Breakpoints

The next question lets you specify whether or not the emulator can use the 68020 BKPT instructions to do software breaks from the user program into the emulation monitor. The **modify sw_breakpoints set** and **run until** commands are disabled if you answer **no** to this question. You should answer **no** only if your target system must use all eight 68020 BKPT instructions.

Enable emulator use of software breakpoints? yes (no)

yes	The emulator software breakpoint functions are enabled.
no	Emulator use of software breakpoints is disabled.

Selecting The Software Breakpoint Instruction Number

The following question lets you specify which of the eight 68020 BKPT instructions the emulator uses to execute software breaks into the emulation monitor.

Note 

If you answered **no** to the previous question, this question will not be displayed on your screen.

Software BKPT instruction number (0..7)? 7 (<number>)

Enabling The Internal 68881 FPU

Note

See chapter 7, "Using Custom Coprocessors", of this manual for detailed information about using coprocessors with the 68020 emulator.

The next question lets you select whether or not the emulator's 68881 FPU is used during emulation. If your target system will eventually have an FPU, but it is currently not available, answering yes to this question enables the emulator's FPU to be used. If your target system has an FPU or if you do not want to use an FPU, answering no to this question disables the emulator's internal FPU.

Enable internal 68881 FPU? no (yes)

yes	The emulator uses the internal 68881 FPU in the emulation pod. If external clock is selected, the maximum clock frequency is 20 MHz. You can use both the internal FPU and other coprocessors during the emulation session. If you are using other coprocessors and want to be able to modify and display coprocessor registers, you must modify the emulation monitor and custom coprocessor format file as described in chapter 7.
no	Use of the internal FPU coprocessor is disabled.

Specifying The FPU Coprocessor ID

Note

If your answered **no** to the previous question, this question will not be displayed on your screen.

This question asks you to specify the FPU coprocessor ID code. The default value is 1. The code is used by the 68020 processor to determine which coprocessor it is accessing for each coprocessor instruction. Note that the value 0 is not available. This value is reserved for the memory management unit (MMU), if used.

FPU coprocessor ID (1..7)? 1

Using Custom Coprocessors

Note

If you answered **yes** to the question "Enable internal 68881 FPU?, this question will not be displayed on your screen. The format file specified for the internal FPU must contain all the information relating to custom coprocessors, if any are used.

The 68020 emulator has the capability to access floating point processors, memory management units, and other coprocessors in your target system. You can both display and modify coprocessor register sets. In order to use custom coprocessors with the emulator, you must provide a custom register format file defining

the coprocessor register set and modify the emulation monitor program as described in chapter 7 of this manual. This must be done prior to modifying the emulation configuration.

Any custom coprocessors? no (yes)

yes	The emulator is enabled to access the custom coprocessors that you have defined in you custom register format file.
no	Use of custom coprocessors is disabled.

Specifying The Custom Coprocessor File

Note 

If you answered **yes** to the question "Enable internal 68881 FPU?" or "Any custom coprocessors?", the following question will be displayed on your screen.

Name of custom register format file?
/usr/hp64000/inst/emul32/0400/0001/custom_spec

The default answer to the question is the name of the custom register format file provided with your emulation software for use with the emulator's internal FPU. If you are using other custom coprocessors, you must enter the full pathname of the custom register format file that you made for these coprocessors.

Modifying a Memory Configuration

When you begin your initial emulation session you must configure (map) the memory space you will be using. The configuration you need is based on your user program requirements and on the configuration of your target system, if one is available. As you progress with your program development, your memory map re-

quirements will probably change. As your requirements change, you will need to modify your configuration file.

The following questions let you review and modify the memory configuration stored in the emulation configuration file.

Modify memory configuration? no (yes)

yes	Allows memory mapping to be modified. The current memory map is displayed. Memory configuration is explained in the following sections.
no	Allows you to skip memory configuration if you do not want to change memory usage. A **no** response causes the memory to be configured as specified by the current emulation configuration file. If **no** is entered, the next question is "Modify emulator pod configuration?".

Break processor on write to ROM? yes (no)

yes	A break to the emulation monitor occurs if the processor attempts to write to a memory location mapped as emulation or target ROM.
no	Breaks are not generated when the processor attempts to write to memory locations mapped as emulation ROM.

If write operations to emulation memory mapped as ROM are attempted during program execution, the contents of emulation memory are not modified. Write operations resulting from emulator commands that modify memory (e.g., load and modify) will modify the contents of emulation memory locations mapped as ROM.

Write operations to target memory mapped as ROM may or may not alter memory contents, depending on your target system hardware.

Mapping Memory

After you answer the question "Break processor on write to ROM?", the emulation memory map is displayed. The 68020 processor memory space required for your applications must be mapped to emulation memory, target memory, or guarded memory. Emulation memory is memory that is physically located in the emulation pod. Target memory is memory that is physically located in your target system. Memory mapped as guarded is memory that, under normal conditions, should not be accessed by your target system. Any reference to the address space mapped as guarded memory will result in an emulation memory break and the display of the error message:

```
STATUS:  68020--Running in monitor      Guarded access  a= <ADDR> (<FC>)
```

where <FC> is a two letter mnemonic describing the function code of <ADDR>.

The memory mapper must be properly programmed to correspond to emulation memory and target system memory resources in order for emulation to work correctly. The memory mapper allows you to divide the processor's address space into blocks that can be individually configured to have any of the following attributes:

- Emulation memory; RAM or ROM; 8-bit, 16-bit, or 32-bit width
- Target memory; RAM or ROM
- Guarded memory

During emulation, the memory mapper monitors the address bus and provides the attributes for the address present at any given time. This information is used by the emulator hardware to control the flow of data and code between the emulation processor and the memory resources.

Memory Map Display Organization. The default memory map display is shown in figure 4-1. Each entry line shows the entry number, address range starting value, address range en-

ding value, function code of the address range, attributes of the entry, and overlay definition. The overlay definition shows the number of the entry being overlaid, and the address in the memory map entry being overlaid that corresponds to the starting address of the overlay entry.

Softkey labels are displayed for the commands available in the memory mapper. You can specify individual map entries, overlay existing map entries, modify existing entries (including the default mapping attributes), delete currently defined entries, or end the map definition session. These commands are described in the following sections.

```
Mapping memory:   Function codes = OFF
  ENTRY    START      END      ATTRIBUTES        OVERLAY
    1          0H  ffffffffH    TARGET RAM                        CPU_SPACE
    2          0H     3ffffH    EMUL RAM [32 bits]




STATUS:    Mapping emulation memory, default mapping:  guarded_________...R....
end

   map    map_over   modify    delete     end     _________  _________  _________
```

Figure 4-1. Default Memory Map Display

Memory Map Definition. The memory map partitions the processor address range into blocks defined as emulation RAM or ROM, target RAM or ROM, or guarded (illegal) space. Each entry defines a particular address range as one of the five possible memory types.

Emulation and target memory entries can be further defined by function code. Emulation memory can also be assigned data port

widths of 8-bit, 16-bit, or 32-bit. Based on the width assignment, emulation memory returns the appropriate DSACK signals to the 68020 processor.

Any address range not defined by an entry is mapped to the memory default. The addresses entered are logical addresses at the appropriate 68020 pins. Adjustments may be necessary if a mapping is used in the target system.

The memory mapper has a resolution of 256 (ffH) bytes. Once the mapper software processes the inputs, the entry range is rounded to integral multiples of 256 bytes. The final range includes all of the specified memory space, plus the remainder of any 256-byte blocks which were partially specified. Any parts of the 68020 address range not defined by an entry are mapped to the memory default.

Note 

If the end address of a specified address range is the same as the first address of a 256-byte memory block (e.g. 100h, xxxxxx00h, etc), the end address value is rounded down one byte (e.g. to 0ffh, xxxxxxffh, etc.)

This can cause a problem if you attempt to specify an address range with the same start and end address corresponding to the first address of a 256-byte memory block. If your enter the command:

map 100h **thru** 100h **emulation ram Return**

the error message **"ERROR: Lower address in range greater than upper address"** is displayed. This command is not allowed by the emulator because the ending address (when rounded down to 0ffh) is less than the starting address (100h).

All emulation memory is displayed and loaded directly by the emulation software by way of the memory port assigned to the host processor. Any attempt by the 68020 CPU to write to memory mapped as emulation ROM will not change the contents of that memory location.

When target memory is specified for a given address range, all memory cycles using that address range access the target system. All memory load and display operations for your target system are done via the emulation monitor.

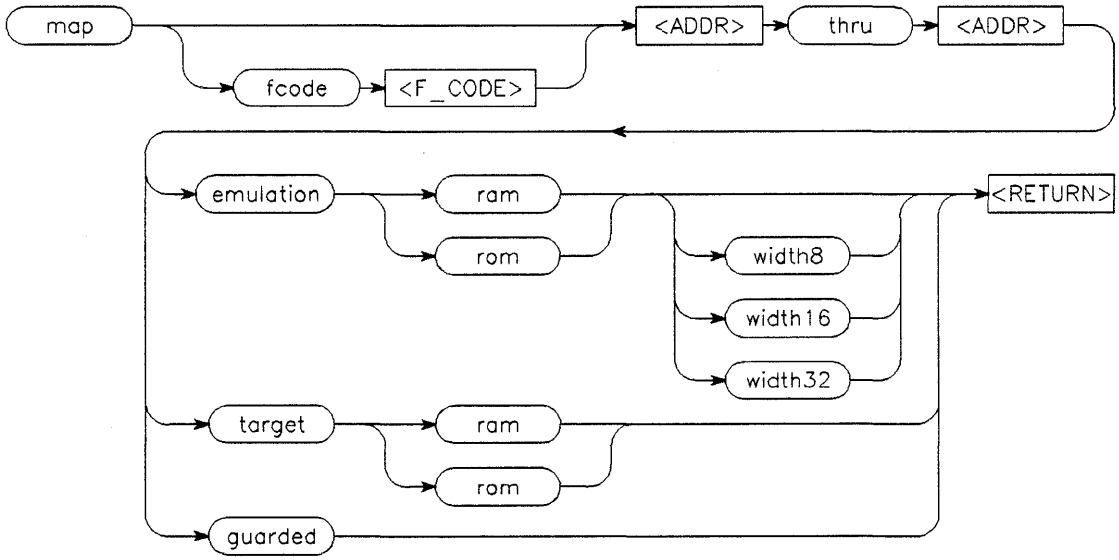
Multiple processor address ranges can be overlaid onto the same physical emulation memory by using the map_overlay command. Overlaying applies only to emulation memory. The emulator has no control over your target system memory resources.

Emulation Monitor Program Memory Requirements. You need to know certain information about the emulation monitor (delivered as part of your emulator software package) prior to linking the monitor program and mapping memory space. Chapter 6 gives a detailed description of the emulation monitor, including memory requirements for the program. Refer to the paragraphs titled "Emulation Monitor Memory Requirements" in chapter 6 for a full description of the emulation monitor memory requirements.

Using The Map Command

All memory map entries are made up of an address range and attributes which specify the type of memory accessed by the specified address range. In addition, a specific function code and address width (port size) can be assigned to a memory map entry. Memory mapping is done using the map command.

Mapper blocks are entered using the following command syntax:



where:

target designates memory supplied by your target system. Mapping an address range to target space requires no emulation memory.

emulation designates memory supplied by the emulation system.

guarded designates an address range which is not expected to be accessed. Any processor access to a location within such a range will result in a break of the program execution. No emulation memory is used when an address range is specified as **guarded**.

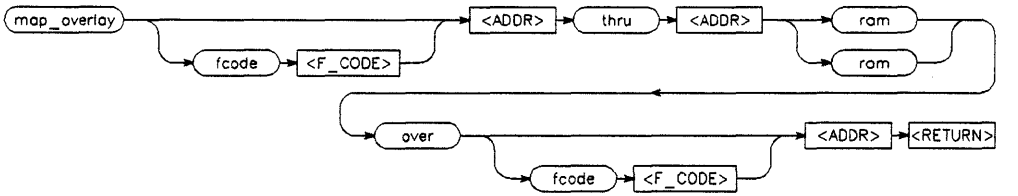
fcode	lets you assign a function code to a memory map entry. The function codes enabled for your particular configuration are displayed on softkeys after you press the fcode key. If you specify **modify defined_codes none**, the fcode attribute is disabled and the softkey is not displayed. You can specify user-defined function codes by typing in the numeric value of the function. See the section in this chapter on the modify defined_codes command for more information on function codes.
rom	designates memory which cannot be modified by the 68020 processor. Emulation memory that is actually RAM but is mapped as ROM performs as ROM during emulation. The host can read and write to ROM.
ram	designates memory which can be read from or written to without restriction.
<ADDR>	defines a bit pattern of up to 32 bits which specifies a particular location in memory. That bit pattern can be entered as a binary, octal, hexadecimal, or decimal number.
width8	defines the memory map entry to be an 8-bit data port.
width16	defines the memory map entry to be a 16-bit data port.
width32	defines the memory map entry to be a 32-bit data port.

The first <ADDR> of a range specification should be the starting address of a block boundary. If an address inside a memory block area is entered, the system converts this address to the starting address of the block prior to its mapping. Leading zeros may be deleted as long as the most significant digit is numeric.

The minimum map entry size is 256 bytes. The maximum size is the number of available blocks.

Using The Map_overlay Command

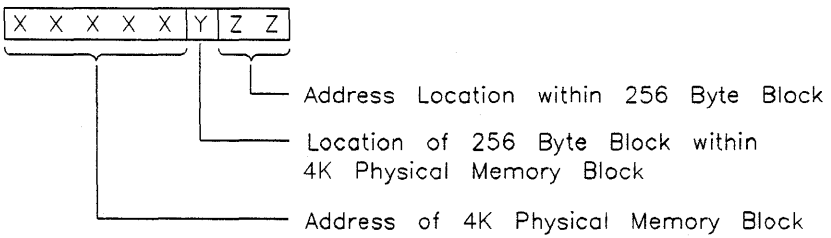
When making a mamory map, you can enter "overlay" addresses in emulation memory hardware blocks. With this feature, you can cause a single block to function as if it were several different blocks, each esponding to a different set of addresses. Memory overlaying applies only to emulation memory. The emulator has no control over target system resources. Map overlays are entered using the following command syntax:



Map_overlay command parameters have the same definitions as those listed for the map command parameters.

There are some restrictions imposed on the map overlay function by the physical structure of emulation memory. Emulation memory is physically made up of 4K byte blocks of memory as shown in figure 4-2. The memory mapper hardware has a resolution of 256 bytes, the minimum map entry size.

When specifying a memory address, the two least significant digits in a hexadecimal address specify the address within the 256 byte entry. The third least significant digit specifies one of the 16 256-byte entries within the 4K byte physical memory block. See the following diagram



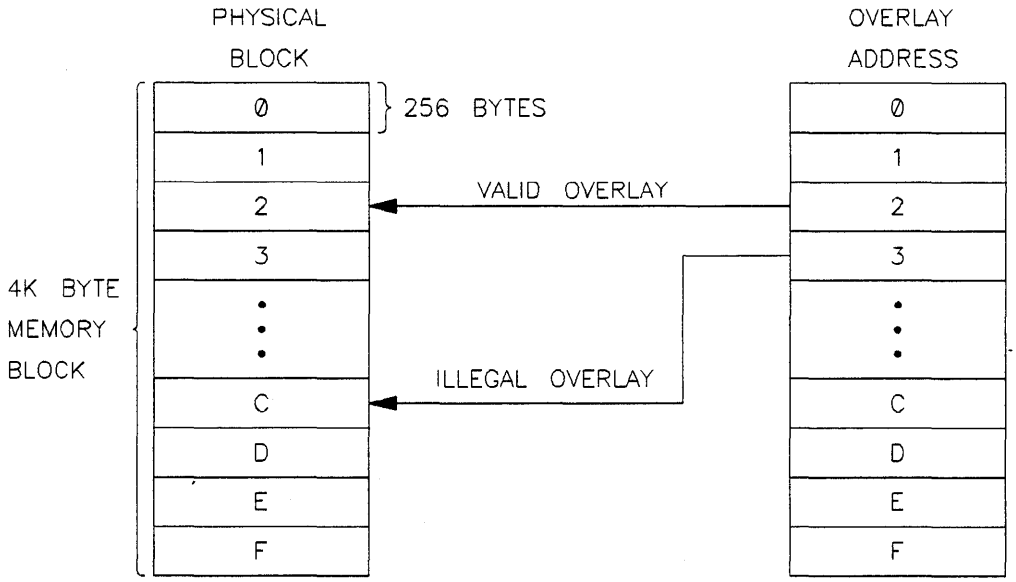


Figure 4-2. Overlay Addressing Within Physical Blocks

When overlaying memory, the address of the memory overlay and the address of the memory location must be mapped to the same 256 byte block in the 4K byte physical memory block, e.g., the third least significant hexadecimal digit in the specified addresses must be identical. For example, the command:

map_overlay fcode SUPER_DATA 0f00f800h **thru** 0f00f8ffh
rom over fcode SUPER_PROG 0002800h **Return**

is a valid command. However the command:

map_overlay fcode SUPER_DATA 0f00f800h **thru** 0f00f8ffh
rom over fcode SUPER_PROG 0002a00h **Return**

is not a valid command. An attempt to execute the last command would cause the error message "Offset for overlay does not match emulation address" to be displayed.

Memory Mapping Example

The following example shows how to map memory in a system made up of a target system with some memory installed and the 68020 emulator. This example shows how to use the the map and map_overlay commands. Before defining the new memory map, delete all entries in the current map. Enter the following commands:

delete all Return
modify defined_codes all Return

The memory map display will appear as shown in figure 4-3. Note that one entry is still displayed. The CPU_SPACE mapping to target RAM cannot be deleted by the user. This address space map is required for vectored exception processing. CPU_SPACE must be mapped to target memory so that vectored exceptions will not interfere with emulation functions.

```
Mapping memory:   Function codes = ON
 ENTRY    START      END       FUNCTION CODES      ATTRIBUTES     OVERLAY
   1          0H  ffffffffH      CPU_SPACE    TARGET RAM




STATUS:    Mapping emulation memory, default mapping:  guarded_________...R....
end
   map    map_over   modify    delete     end     _________ _________ _________
```

Figure 4-3. Sample Overlay Mapping #1

Type the following entries into the memory map.

map fcode USER_DATA 0 thru 0ffffh **emulation ram Return**

map fcode USER_PROG 18000000h **thru** 1800ffffh **emulation rom Return**

map fcode SUPER_DATA 0 **thru** 3ffh **target rom Return**

map fcode SUPER_PROG 0 **thru** 3ffh **target rom Return**

map fcode SUPER_PROG 0f000000h **thru** 0f000ffffh **emulation ram Return**

map_overlay fcode SUPER_DATA 0f000000h **thru** 0f000ffffh **ram over fcode SUPER_PROG** 0f000000h **Return**

```
Mapping memory:   Function codes = ON
  ENTRY     START        END       FUNCTION CODES     ATTRIBUTES      OVERLAY
    1          0H       ffffH       USER_DATA    EMUL RAM [32 bits]
    2   18000000H   1800ffffH       USER_PROG    EMUL ROM [32 bits]
    3          0H       3ffH      SUPER_DATA    TARGET ROM
    4   f000000H   f000fffH      SUPER_DATA    EMUL RAM [32 bits] f000000h (6)
    5          0H       3ffH      SUPER_PROG    TARGET ROM
    6   f000000H   f000fffH      SUPER_PROG    EMUL RAM [32 bits] f000000H
    7          0H  ffffffffH       CPU_SPACE    TARGET RAM




STATUS:    Mapping emulation memory, default mapping:  guarded_________...R....
 end

   map    map_over   modify    delete     end     _________  _________  _________
```

Figure 4-4. Sample Overlay Mapping #2

The memory map resulting from these commands is shown in figure 4-4. The entries in the memory map correspond to the following address spaces:

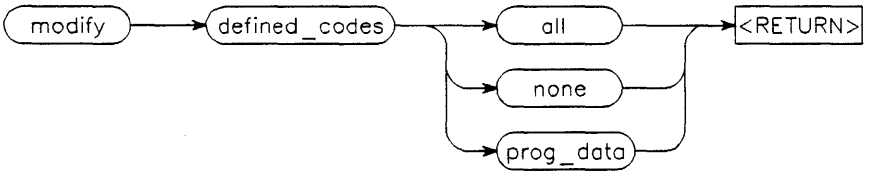
1. User application data space
2. User application program space
3. Exception vector table space
4. Emulation monitor data space
5. Exception vector table space
6. Emulation monitor program space
7. CPU space

The emulation monitor data space (entry 4) has been overlaid onto the emulation monitor program space, This enables the 68020 processor to access data locations in the emulation monitor. The overlay is indicated in the OVERLAY column of the memory map display for entry 4. The "(6)" indicates that entry 4 is overlaid onto entry 6. The address f000000H is the address in entry 6 that corresponds to the starting address of entry 4. This memory map shows you a typical 68020 memory map.

Using The Modify Command

The modify command lets you modify the memory map. The modify defined_codes command lets you selectively enable or disable the 68020 function code signals (FC0 through FC2). The modify <ENTRY> command lets you modify the range, attributes, fcode, and overlay parameters of a memory map entry. The modify default command lets you change the default memory parameters.

Modify Defined_Codes. The modify defined_codes command lets you selectively enable or disable the 68020 function code signals. The command syntax is shown in the following diagram:



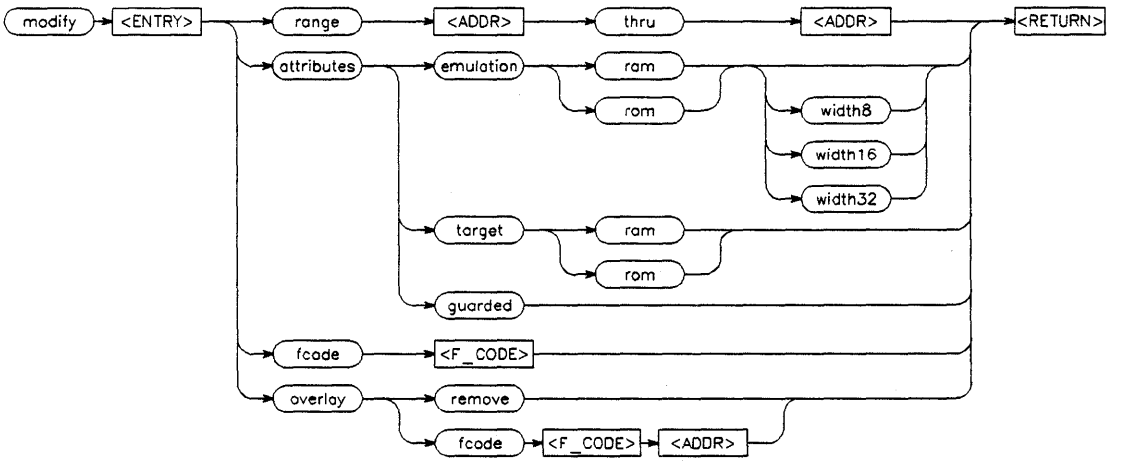
where:

all enables the memory mapper to use all three function code lines (FC0 through FC2) in mapping memory. If all is selected, you can specify any of the eight function code states except CPU_SPACE. The function codes SUPER_PROG, SUPER_DATA, USER_PROG, AND USER_DATA can be entered from softkeys. The remaining function codes must be entered as numeric values. Function code 3 is user definable. Function codes 0 and 4 are reserved for use by the processor manufacturer. Function code 7 specifies CPU address space. If you enter fcode 3, **USER_RSVD** is displayed in the FUNCTION CODES column of the memory display. If you enter fcode 0 or 4, **MOT_RSVD** is displayed in the FUNCTION CODES column.

none disables all three function code lines. when none is selected, the emulator memory mapper ignores the function code lines and monitors only the 32-bit address bus during emulation. With none selected, the fcode parameters are not available in the emulation commands. The FUNCTION CODES column is deleted from the memory map display.

prog_data enables the memory mapper to monitor only function code lines FC1 and FC0. These lines determine whether address space is defined to be program address space or data address space. With prog_data selected, you can only specify the function code to be program or data address space. The function codes PROG or DATA can be entered from softkeys.

Modify <ENTRY>. The modify <ENTRY> command lets you modify the range, attributes, fcode, and overlay parameters of a existing memory map entry. The command syntax is shown in the following diagram:



where:

range lets you specify a new range for the memory map entry (<ADDR> **thru** <ADDR>).

attributes lets you change the entry to:

emulation memory, RAM or ROM, with a data port width of 8-bits, 16-bits, or 32 bits

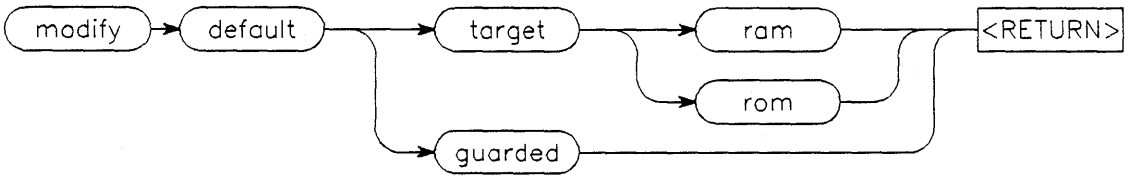
target memory, RAM or ROM

guarded

fcode lets you modify the function code address mapping for the entry. The selections available to you depend the definition of the defined_codes parameter.

overlay lets you remove an overlay from an entry, e.g., the entry is converted to the physical address corresponding to address specified in the entry, or it lets you change the function code or address range of the address space being overlaid.

Modify Default. Any address ranges which are not mapped when the mapping session is terminated are assigned the memory attribute specified as the default. The default attribute can be set up to be target RAM, target ROM, or guarded by using the modify default command. Initially, the system assigns all unmapped memory to guarded memory. The command syntax is shown in the following diagram:



where:

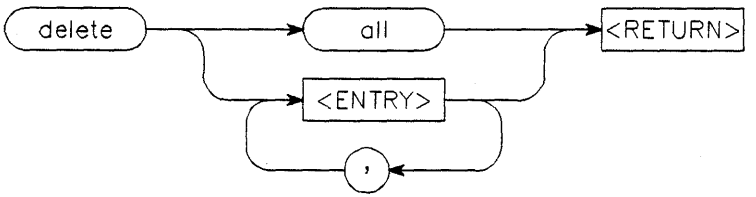
target designates memory supplied by your target system.

guarded designates an address range which is not expected to be accessed. Any processor ac-

cess to a location within such a range will result in a break of the program execution.

Deleting Memory Map Entries

Any one or all of the memory map entries can be removed by using the delete command with the exception of the default CPU_SPACE entry. The syntax for the delete command is shown in the following diagram:



Ending The Mapping Session

The memory map configuration session is exited by pressing the **end** softkey followed by **Return**.

Modifying The Emulation Pod Configuration

The following question asks you whether or not you want to modify the current emulation pod configuration.

Modify emulator pod configuration? no (yes)

no The emulation pod configuration questions are skipped and the emulation module uses the current pod configuration. The emulator will skip to the end of the configuration session and ask you for the emulation configuration file name. The default pod configuration is as follows:

Enable DMA transfers yes
Enable DMA transfers into emulation memory no
CPU clock source internal
Interlock emulation memory DSACK with user DSACK no
Enable emulator use of INT7 yes
Enable target IPEND line during emulator breaks no
Block target BERR during emulation memory cycles yes
Enable on-chip cache no

yes	You must answer the following emulator pod configuration questions in order to reconfigure the emulator pod.

Enable DMA transfers? no (yes)

no	Bus requests are blocked to the processor and the analyzer does not capture DMA activity. The processor ignores the BR and BGACK input signals and does not respond with BG.
yes	Bus requests are admitted to the processor. If the AS, address, and data lines are active at the processor pins during DMA cycles, the analyzer will capture those states. The processor responds normally to the assertion of the BR (Bus Request) and BGACK (Bus Grant ACKnowledge) signals.

Enable DMA transfers into emulation memory? no (yes)

Note 

If you answered no to the previous question, this question is not displayed on your screen.

---

no	DMA transfers to memory addresses mapped as emulation memory are disabled.
yes	DMA transfers to memory addresses mapped as emulation memory are enabled. The DMA device must generate all required control signals (AS, DS, R/W, SIZ, etc.) and meet the 68020 timing specifications.

CPU clock source? internal (external)

internal	The internal clock must be used when the emulator is running out-of-circuit, i.e., with no target system. You may use the internal clock when running in-circuit emulation, i.e., with your target system connected to the emulator if the target system does use its own CPU clock. You do not need to disable your target system clock when the internal clock is selected. However, the internal clock is not driven out to the target system. Therefore, if your target system needs a clock to operate, the external clock must be selected. The internal CPU clock runs at 16.667 MHz.
external	An external clock is normally used when the emulator is connected to your target system. Your target system clock must meet the specifications for the CPU CLK input to the microprocessor in order to be reliably used with the 68020 emulator. The emulator operates with clock rates up to 25 MHz, except when the internal FPU is enabled. Then, the maximum clock rate is 20 MHz.

CPU clock rate greater than 20MHz? no (yes)

Note 

If you answered **internal** to the previous question, this question is not displayed on your screen.

---

no	If the external clock rate is less than or equal to 20MHz, all emulation memory accesses will occur with no wait-states.
yes	If the external clock rate is greater than 20MHz, one wait-state will be inserted for emulation memory and target memory accesses.

Add wait states to target accesses? no (yes)

no	The 68020 emulator will not force wait states for target memory accesses. If the target system automatically generates wait states, these wait states will still be present for target memory accesses
yes	The 68020 emulator will ensure that there is at least one wait state for target memory accesses. If the target system already meets this requirement, the emulator will have no effect on accesses to target memory.

Interlock emulation memory DSACK with user DSACK? no (yes)

no	DSACKs for emulation memory accesses are generated by the emulator, according to the mapped size of that memory. Target system DSACKs are used for all target memory accesses.

yes The target system DSACKs are used for all memory accesses. You must ensure that the mapped size of emulation memory matches the target system DSACK signals. This feature can be used to keep the emulator synchronized to the target system while accessing emulation memory.

In most cases, DSACKS should be interlocked when the emulator is plugged into a target system. Refer to chapter 5, "Using the Emulator", for a detailed discussion of emulation and target sytem DSACK signals. Note that the emulator does not interlock DSACK signals during the level 7 interrupt jamming process that occurs during the emulation break function.

Enable emulator use of INT7? yes (no)

The emulation break function uses the level 7 interrupt autovector (INT7) processor resource to force the user program to be interrupted and the emulation monitor program to be entered. This question lets you enable or disable the emulation break function, as required for your target system. If your target system cannot share INT7 with the emulator, you need to answer no to this question.

yes All selected emulation functions are available.

no All emulation break signals to the processor are disabled. The only ways to enter the monitor program are:

- user program jumps to the monitor
- executed exception vector points to the monitor
- software breakpoint is executed
- reset command with reset-to-monitor function enabled

Enable target IPEND line during emulator breaks? no (yes)

no	The interrupt pending signal (IPEND) is blocked (driven high) for ALL interrupts, both emulator and target system generated interrupts.
yes	Any interrupt sends the interrupt pending signal (IPEND) to the target system.

Block target BERR during emulation memory cycles? yes (no)

yes	Bus errors (BERR) that occur during emulation memory cycles are blocked. This allows the monitor or other user program to run in a memory space not usually allowed by the target system hardware.
no	All bus error signals (BERR) are transmitted to the processor.

Enable on-chip cache? no (yes)

no	The processor is forced to always access external memory. You must answer no in order to use all of the analysis features.
yes	The processor executes the instruction in the cache, if the required word is stored there. A yes answer improves system performance but much analysis capability is lost. The enable (E) bit of the CPU CACR register must be set by the target software for the cache to be enabled.

Refer to chapter 5, "Using the Emulator", for more information regarding the on-chip cache.

Configuring Simulated I/O

The simulated I/O subsystem must be set up by answering a series of configuration questions. These questions deal with enabling simulated I/O, setting the control addresses, and defining files used for standard I/O.

Configure Simulated I/O? no (yes)

Answering yes to this question causes a series of simulated I/O questions to be asked. For information on how to answer these questions to configure your system, refer to chapter 8 of this manual. For additional information about simulated I/O, refer to the *Simulated I/O Reference Manual.*

Answering no to this question bypasses all other simulated I/O questions.

Configuring Simulated Interrupts

Simulated interrupts are enabled by answering a series of configuration questions.

Modify simulated interrupt configuration? no (yes)

If you answer yes, the simulated interrupts questions will be asked. If you answer no, the questions will be skipped. Simulated interrupts enable you to write and test software which depends upon the occurrence of preemptive interrupts using an emulator that is out of circuit. Information describing how to configure your system for simulated interrupts is contained in chapter 8 of this manual.

Naming The Configuration File

This question lets you name an emulation configuration file containing the emulation configuration information you have just entered. The configuration file is stored on disc and can be called up for use during a future emulation session.

Configuration file name?

Type in the filename you want and press **Return**.

If you press **Return** without entering a name, the current emulation session will be configured as you specified in your answers and the information will be saved as the new default configuration of the emulator. To restore the original default file provided with the emulation software, you must reinitialize the HP 64120A Cardcage.

Note If you assign a new name to the configuration file and you are using a command file to enter your emulation session, remember to modify your command file to change the name of your emulation configuration file (refer to the *HP 64000-UX User's Guide* for more information relating to command files).

Note Emulator configuration files are slot dependent. Use of a given configuration file on one emulator and subsequent reuse on an emulator in another cardcage slot will result in the message "Bad Module File". This message indicates that the configuration file specified was not associated with the current emulator. The message is displayed as a warning only. The emulator software will automatically rebuild the configuration file with correct cardcage slot information for the current emulator.

Configuration Switches

See Figure 4-1. There are three switches located on the emulation processor board in the pod. They allow you to select some infrequently changed hardware options. These switches are described in in the following paragraphs:

C1 Selects either a buffered or unbuffered clock.

1. Default setting: The target clock signal is buffered by a 74F241 before it is sent to the 68020 microprocessor in the

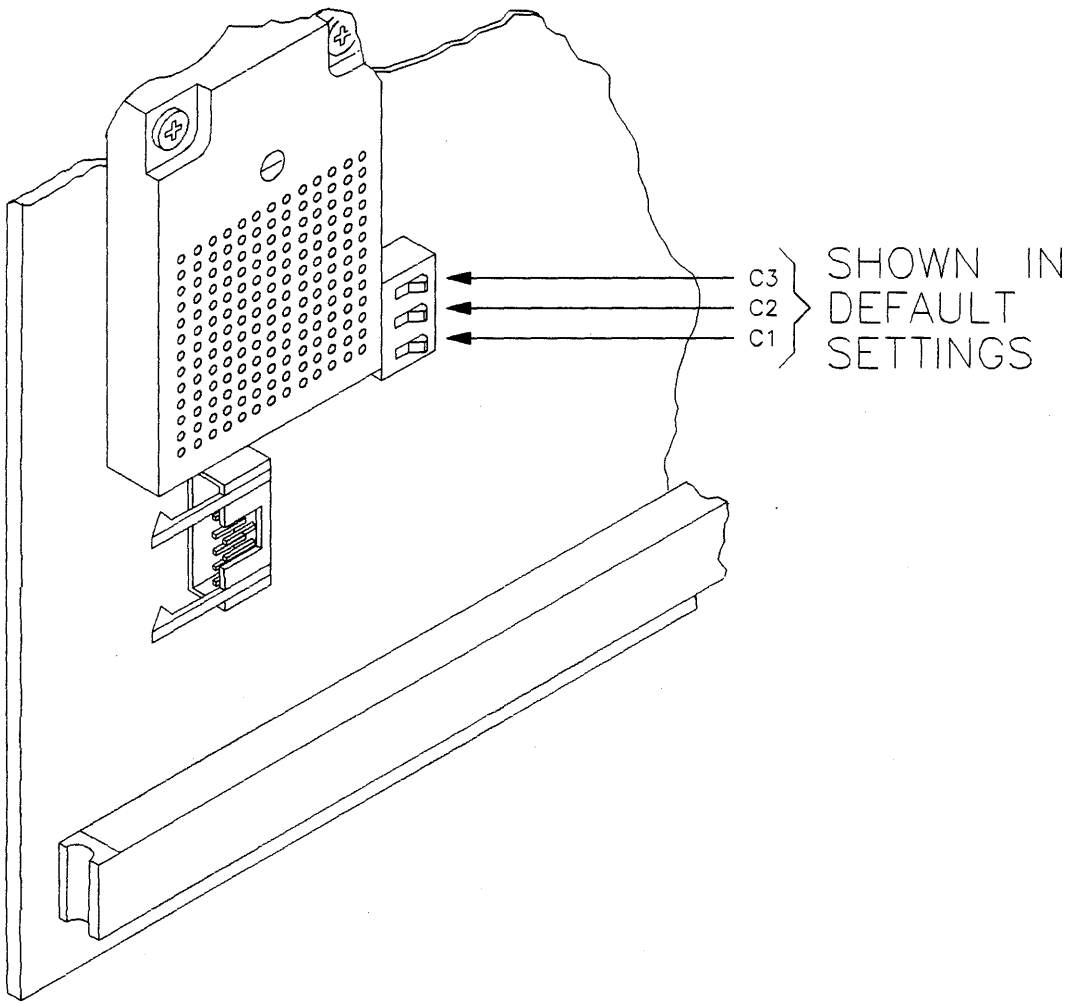


Figure 4-5. Setting Configuration Switches

emulation pod. The shape of the clock signal is improved with this buffering technique.

2. Switched Setting: The target clock signal is sent directly to the 68020 microprocessor in the pod. The clock signal is not buffered by the emulator. Use this setting only if the clock skew (which results from buffering the signal) causes timing problems when plugged into your target system.

C2 (DSACK0) And C3 (DSACK1)

Selects either buffered or unbuffered DSACK signals.

1. Default Setting: This imputs the target DSACK signals to a PAL before they are sent to the 68020 microprocessor in the pod. These switches must be set to default during normal emulator operations.

2. Switched Setting: Use this setting only as a troublesetting tool for trying to isolate target plug-in problems. In this mode, the target system DSACKs are sent directly to the 68020 microprocessor in the pod, bypassing all of the emulator DSACK logic. This improves the emulator timing specification pertaining to the DSACK signals (spec. number 28), but reduces other emulator functionality. Functions that use the emulation monitor or the internal FPU are not operational in this mode! The emulator basically functions only as a preprocessor. All memory should be mapped to target. Only the analyzer features should be used in this mode.

Notes

5

Using The Emulator

Overview

This chapter provides information on the appropriate use of the following emulator and processor features when the emulator is used with a target system (in-circuit emulation):

- Installing emulation software updates
- Emulation and target system $\overline{\text{DSACK}}$ signals
- Vector base register
- The internal 68020 cache
- Using function codes for displaying and modifying reserved address space
- Enabling/disabling the bus error signal ($\overline{\text{BERR}}$)
- Using DMA
- Using the run from ... until command
- Using the emulation monitor
- Target systems with memory management units (MMU's)
- Memory access timing issues
- Loading absolute files.

Read this chapter before attempting to operate the emulator with your target system.

Installing 68020 Emulation Software Updates

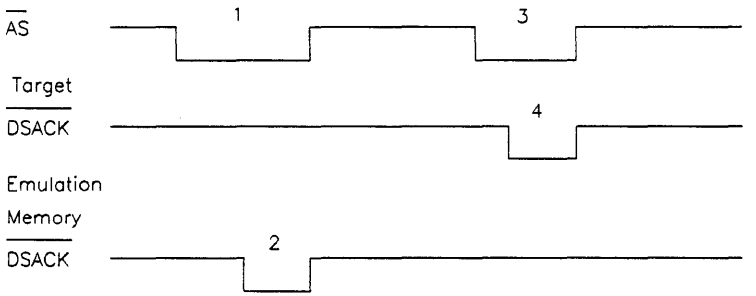
After installing a new copy of the 68020 Emulation Software on a system, cycle the power off and then back on for all HP 64120 cardcages containing 68020 emulators. This updates and initializes all emulation software data structures.

When installing a different revision of the 68020 emulator software, remake all existing configuration files. Configuration file names are suffixed by ".EA" and ".EB". The easiest method to make a new configuration file is to delete the **.EB** file and then reload the configuration file. The system will take the parameters specified in the **.EA** file and make a new **.EB** file that is compatible with the new emulation software.

Emulation And Target System DSACK Signals

Interlocking Emulation Memory DSACK and Target DSACK Signals

If your target system memory requires wait states, you should interlock the emulation memory DSACK signal with the target system DSACK signal. This causes accesses to emulation memory and accesses to target memory to properly reflect system performance when the emulator is removed. Since emulation memory has zero wait states up to 20 MHz, accesses are much faster than when the emulator is removed from the target system and the same accesses are made to target system memory that has wait states.



1 An access to emulation memory.
2 Emulation memory DSACKs terminate cycle properly.
3 Access to target memory.
4 Target DSACKs from emulation memory accesses (1) prematurely terminate the cycle before correct data is available from target memory.

Figure 5-1. Memory Access Timing, No DSACK Interlock

Note

When operating the emulator at 25 MHz, one wait state will be added **EVEN** if the target system responded with a zero-wait-state termination during interlock operation.

If target system memory requires wait states, the first target memory access after an emulation memory access may fail if DSACKs are not interlocked. See the timing diagram in figure 5-1.

The following rules should be used to determine whether or not to interlock DSACK signals.

1. If the target system generates DSACK signals for all emulation memory address ranges, DSACK signals should be interlocked.

2. If the target system does not generate DSACK signals for a range of emulation memory, DSACK signals must not be interlocked.

3. If there is no target system (i.e. out-of-circuit emulation), DSACK signals cannot be interlocked.

To interlock emulation memory DSACK signals with target system DSACK signals, answer yes to the emulation configuration question **"Interlock emulation memory DSACK with user DSACK?"**.

DSACK Signal Problems In Target Systems

Many target systems violate 68020 DSACK signal specifications. These violations are usually marginally acceptable to the 68020 CPU in the target system, but cause problems when the emulator is plugged in. These specification violations usually result in improper data fetches from memory and cause target system failure with the emulator installed.

Use Of Open Collector Drivers

One of the most common problems is associated with the use of open-collector drivers on the DSACK lines. DSACK lines often have pullup resistors that pull the DSACK signals high at the termination of a memory cycle. Improper values for pullup resistors can cause DSACK signals not to be pulled up fast enough and may interfere with the next cycle. This occurs when the pullup resistor value is too large to return DSACK to a proper high level before the next cycle begins. In this case, the still low DSACK signal causes a premature termination of the second cycle, resulting in improper data fetches by the CPU.

Early Removal Of DSACK Signals

Some target system designs do not adhere to the 68020 specification which states that the DSACK signals must not be removed prior to the negation (low to high transition) of the address strobe at the end of a cycle. In the simplest case, this results in "no DSACK" messages appearing in the tracelist, which in turn causes inverse assembly failure. More seriously, the emulator may completely malfunction depending on how early the DSACK signal is removed prior to address strobe transition.

Isolating The DSACK Problem

If you suspect that your target system may HAVE either of the preceding problems, use a timing analyzer to help isolate the problem. Take a trace of the CPU clock, address strobe, data strobe, and the DSACK signals during the failing cycle (use the BNC's on the back of the HP 64120 cardcage to drive the trigger, if possible). Examine the results and compare your findings to the electrical specifications of the 68020 processor and the HP 64410SC/SD emulator.

Note 

HP 64120 BNC port operation is available only with HP 64410SC/SD software version 1.10 and later versions.

Using The Vector Base Register

The 68020 CPU gets exception vectors from the exception vector table located at the address contained in the Vector Base Register (VBR).

The 68020 emulator uses a jamming technique for breaks and software breakpoints. Therefore, the value of the VBR is not needed to perform most monitor functions. This implies that the vector table may be located anywhere without adversely affecting emulator operation.

The single-step feature does require the use of the trace exception vector (VBR + 24H). If the single-step feature is to be used, you must make sure that the trace exception vector always points to the monitor (MONITOR_ENTRY).

The monitor can handle various exceptions by displaying a status message, entering a loop within the monitor, and then waiting for user intervention. These exceptions include Bus Error, Address Error, Divide by zero, etc. If you use these exceptions, you must maintain the exception vector table so that the vectors in use always point to the appropriate monitor location.

Using The Internal 68020 Cache

Using the internal 68020 cache affects several functions of the 68020 emulator. The following sections discuss use of the internal cache and its effect on emulator operation.

Cache Control

When the emulator is operating out-of-circuit, the "Enable Cache?" configuration question has a different interpretation than when plugged into a target system. When using the emulator out-of-circuit, a "yes" answer to the "Enable Cache?" configuration question forces the $\overline{\text{CDIS}}$ signal high within the pod. When using the emulator in-circuit, a "yes" answer connects the target system $\overline{\text{CDIS}}$ signal to the emulator CPU's $\overline{\text{CDIS}}$ input, allowing the emulator to track target system $\overline{\text{CDIS}}$. In both cases, a "no" answer forces $\overline{\text{CDIS}}$ low within the emulator.

Recall also that the target system $\overline{\text{CDIS}}$ must be high, and bit zero of the Cache Control Register must be set to 1 for the cache to be enabled.

If the target system uses the internal 68020 cache, the cache must be enabled by answering "yes" to the "Enable Internal Cache?" configuration question.

When the $\overline{\text{CDIS}}$ signal from the target system is set to 1, the cache still is not enabled until bit 0 of the cache control register (CACR) is set to 1, as shown in the following example:

```
MOVEQ.L  #1,D0
MOVEC    D0,CACR          ;software enable cache
```

Enabling the cache affects analysis trigger, store, count, and Global Context functions. Additionally, some program read states may be missing from the trace list.

The cache is not frozen on entry to the monitor. This results in overwriting the cache contents.

If a breakpoint is set for an address currently contained in cache, the breakpoint will not be recognized until the CPU fetches from that address in main memory again. The run until command is similarly affected since breakpoints are used in the command implementation.

Analysis with Cache

The 32-bit internal analyzer can capture any cycle that occurs external to the 68020 CPU. When cache is enabled, program read cycles may occur only internal to the CPU. This is the general case with tight program loops and with high performance code segments that are frequently locked in cache. Since the analyzer cannot capture internal cycles, it has no way to display these cycles in the tracelist. This can result in missing trace data and high-level source lines, and even improper disassembly. The analyzer will also miss the occurrence of trigger, store, count, sequence or context patterns if they occur only as internal cycles.

With cache enabled, all non-program-read cycles occur externally, since the 68020 cache is implemented as an "instruction only" cache. In general, any program segment that executes from cache will generate some external cycles, the major exception being timing loops. In these cases, you may be able to select trigger and store patterns that correspond to external cycles. If no external cycles are normally generated, you may be able to place "markers" in the cached code such that the code will generate an external cycle for analysis purposes when executed.

Since the analyzer contains a high precision cycle-to-cycle timer, you can usually examine the tracelist to determine where cache execution occurred.

Using Breakpoints With Cache Enabled

You may see situations where breakpoints do not appear to be functioning properly when the cache is enabled. This can happen when you are using the "run until" command as well as breakpoint commands.

Consider the following segment of code (a simple software timing loop), and assume that the cache is enabled:

```
Address            Code

1000:    RELOOP    NOP
1002:              NOP
1004:              NOP
1006:              NOP
1008:              NOP
100A:              SUBQ.L    #1,D0       ; decrement loop counter
100C:              BNE       RELOOP      ; reloop if not 0
```

Because cache is enabled, no external memory cycles are generated for addresses 1000H thru 100CH after their initial load into cache. Breakpoints set at any cache resident address may never be encountered. This situation occurs when the CPU does not generate an external program read cycle to memory and therefore never "sees" the breakpoint that was set.

Target Memory Breakpoints

Breakpoints set in target system memory differ from those set in emulation memory. If the breakpoint address is mapped to target system memory, the monitor must intervene in order to set the breakpoint. Execution of the monitor overwrites cache locations previously occupied by the user program. When the emulation monitor is exited, the user program is fetched again from memory, breakpoint included. This results in normal breakpoint behavior.

Emulation Memory Breakpoints

This problem is worse when the breakpoint address is mapped to emulation memory. Due to the dual-port nature of the memory system, the host sets breakpoints in emulation memory without requiring execution of the emulation monitor. In this case, the mechanism of setting breakpoints does not clear cache and force a refetch of the newly specified breakpoint.

For breakpoints to function properly out of emulation memory, you need to clear the cache before setting or resetting the breakpoint. Do the following steps before setting a breakpoint:

1. Break to the emulation monitor program.
2. Display CPU registers.
3. Modify CACR bit C to 1 and then to 0.
4. Set the breakpoint or enter the run until command.
5. Exit the monitor by executing a run command.

When the breakpoint is hit, you can remove it from cache by adding 68020 instructions to the emulation monitor that will set and clear the CACR C bit.

The preceding comments apply to setting software breakpoints as well as disabling software breakpoints.

Using Function Codes For Displaying And Modifying Reserved Address Space

When the use of function codes is enabled during a memory mapping session, the display and modify commands use the function codes specified in the command. When function codes are disabled, function code 0 is used for all memory reference commands.

Some target systems do not use function codes to differentiate between user and supervisor space or program and data space, but do decode the "reserved" address spaces (function codes 0, 3 and 4) to generate interrupts or inhibit DSACK generators. In these cases, the emulation monitor may be customized to allow the use of a non-zero function code for the display, modify, load, and store emulator commands.

This modification requires changing two assembly statements in the monitor "COPY" routine as shown in the following listing:

```
      ********************************************************
      *
      *     COMMANDS 3 & 4 ..... ACCESS USER MEMORY
      *
      ********************************************************

      COPY
      MOVE.W    BYTE_COUNT,D3
      MOVEA.L   SRC_ADDR,A0      ; GET MEMORY SOURCE ADDRESS.
      MOVEA.L   DST_ADDR,A1      ; GET MEMORY DESTINATION ADDRESS.
>>> ; MOVE.W    DST_FC,D2        ; GET DESTINATION FUNCTION CODE
>>>   move.w    #5,d2            ; force supr data function code
      MOVEC     D2,DFC
>>> ; MOVE.W    SRC_FC,D2        ; GET SOURCE FUNCTION CODE
>>>   move.w    #2,d2            ; force user data function code
      MOVEC     D2,SFC
      CMPI.L    #3,MON_COMMAND   ; IS THIS A READ_MEMORY COMMAND?
      .
      .
      .
```

Modifications to the emulation monitor code for non-zero function code access to target system memory include adding the two new source lines shown in lower-case and commenting out 2 lines as shown in the listing. The added and modified lines are indicated by arrows (>>>).

Enabling/disabling BERR

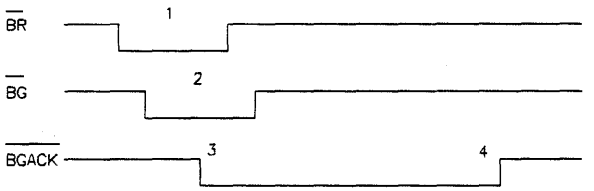
The 68020 emulator allows the bus error ($\overline{\text{BERR}}$) signal to be received or not received during accesses to emulation memory.

If the target system generates bus errors for emulation memory address ranges, the reception of $\overline{\text{BERR}}$ should be disabled. This would normally occur if $\overline{\text{DSACK}}$s are not generated for emulation memory accesses.

If the target system generates $\overline{\text{DSACK}}$s for emulation memory accesses, then it probably does not generate $\overline{\text{BERR}}$ for these cycles. In this case, $\overline{\text{BERR}}$ actually indicates a failure, and should be enabled in the emulator.

Using DMA

If any devices share the 68020 bus and are able to perform DMA, then DMA should normally be enabled. This enables the CPU to receive the Bus Request (BR) signal, generate a Bus Grant (BG) response signal, and receive the Bus Grant Acknowledge (BGACK) response from the bus requester. The handshake sequence for DMA transfers is shown in figure 5-2.



1) External device requests the bus.
2) The 68020 indicates that the bus will be granted.
3) External device indicates that the bus is in use.
4) External device relinquishes the bus.

Figure 5-2. DMA Bus Request/Bus Grant Timing

If DMA is disabled, the CPU will not receive the bus request signal, and will not allow DMA cycles. This would be desirable in order to characterize system performance in a situation where DMA could not occur.

If the target has an MMU (any type), bus arbitration will normally occur during some types of address translation cycles. Thus, if an MMU is present, DMA should usually be enabled for proper system operation.

The user who has enabled DMA has a secondary option of enabling or disabling DMA to/from emulation memory.

If DMA to emulation memory is enabled, the DMA hardware has access to read from or write to emulation memory. If DSACKs are interlocked, the DSACKs for these accesses are supplied by the target system. The DMA master must generate cycles that conform to 68020 timing requirements.

If DSACKs are NOT interlocked, then no DSACKs are returned to the target system. This would cause the DMA hardware to hang if DSACKs are required for cycle termination. To provide maximum flexibility, the emulator makes DSACK signals available on an external connector, and in addition presents a signal differentiating target and emulation memory cycles:

External Connector Label	Meaning
CD0	CPU DSACK0
CD1	CPU DSACK1
USR	USER (TARGET) MEM = 0, EMUL MEM = 1

CD0-CD1 are DSACK indications from emulation memory when USR = 1. If USR = 0, CD0-CD1 reflect target memory DSACKs. Using these signals, the circuit in figure 5-3 is needed to perform DMA to/from emulation memory under the following conditions:

1. DMA enabled (DMA to emulation memory enabled or disabled).
2. Interlocking DSACKs is disabled.
3. DMA hardware requires DSACKs.
4. Target system does not generate DSACKs for emulation memory accesses.

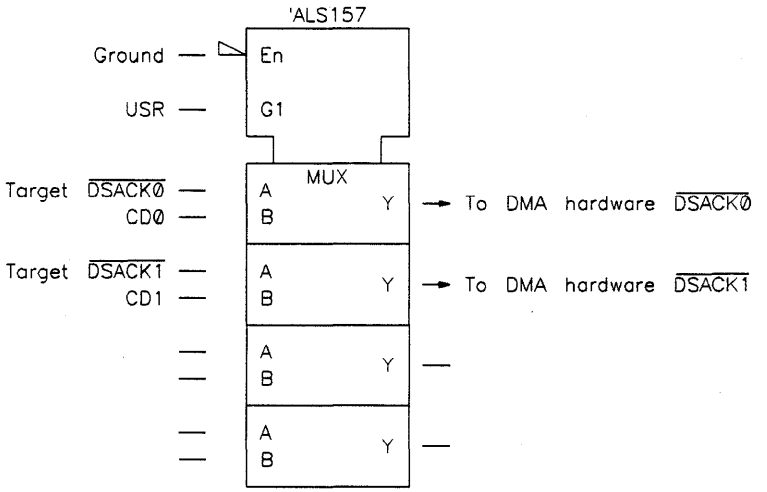
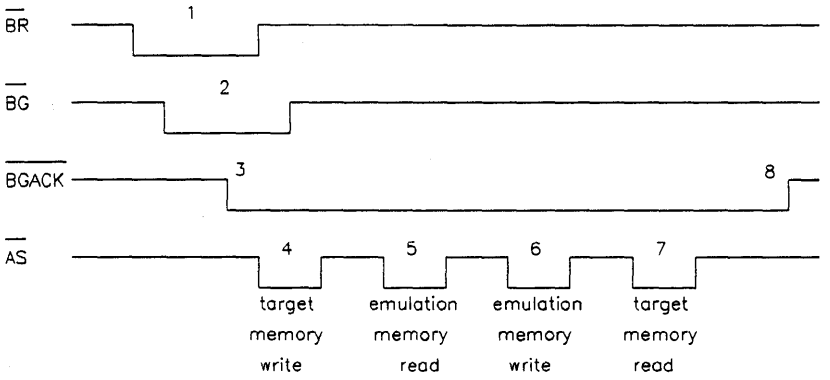


Figure 5-3. Circuit For DMA Transfers

If the option to DMA to/from emulation memory has been DISABLED, the DMA cycle will still be allowed to occur, but no information will be written to, or read from emulation memory. See the timing diagram in figure 5-4.



1 DMA device requests the bus.
2 The 68020 indicates that bus will be relinquished.
3 DMA device indicates that the bus is in use.
4 DMA device generates a write with a target memory address. This cycle occurs normally.
5 DMA device generates a read with an emulation memory address. This cycle does not return valid data since DMA to emulation memory is disabled.
6 DMA device generates a write with an emulation memory address. This cycle does not modify emulation memory since DMA to emulation memory is disabled.
7 DMA device generates a read with a target memory address. This cycle occurs normally.
8 DMA transaction is complete.

Figure 5-4. DMA Timing Diagram, DMA Disabled

Using The Run From ... Until Command

The run command must be used properly to avoid serious, stack related problems in the software being executed.

One of the main causes of target system "failure" while using the run command is the stack not being setup and/or restored properly by the software being executed. One common situation is for parameters to be placed on the stack prior to calling a procedure. (Parameter stacking code including the actual procedure call is usually referred to as the "calling sequence.") Assume for example that a procedure, PROC1 expects the stack frame shown in figure 5-5.

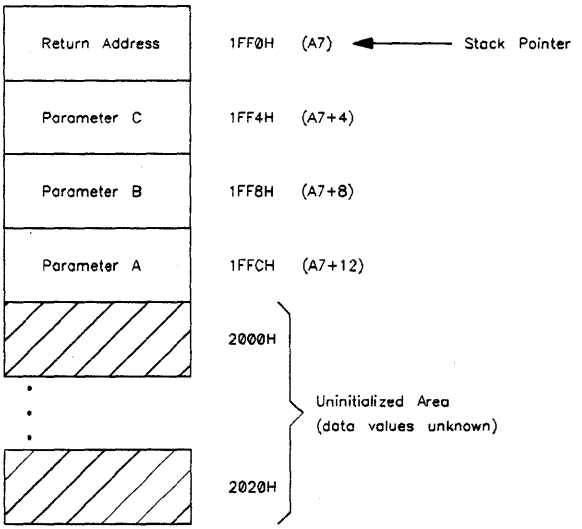


Figure 5-5. Example Stack Frame

Often, PROC1 will access the stacked parameters by referencing parameter requests to the stack pointer. This means that parameter "A" can be found at address A7 + 12, parameter "B" at address A7 + 8, etc.

If the parameters are not stacked, and/or the return address is not present, then the usual parameter references A7 + 12, A7 + 8, etc. may reference uninitialized stack areas. Also, the return address used by PROC1 will be incorrect. This will usually result in a software failure both within PROC1 (because the parameter values are wrong) and on exit from PROC1 (because the return address was not set properly). Depending on emulator memory mapping, the "stack" areas referenced by A7 + 12, etc. may actually fall within guarded memory area, resulting in a guarded memory access message.

Executing the command **"run from** PROC1" prior to stacking the parameters and setting the return address is one case where this could happen. Problems also occur if a "run from <address>" command is executed and CPU registers, or memory locations are not properly initialized for the code to be executed at <address>.

Using the command **"run until"** can also cause problems. This case is different from the "run from" case in that software problems may occur on a subsequent "run" command after the "until" condition is satisfied. If a "run" command is executed after executing the "until" breakpoint, no problems should result, because the CPU will continue executing the user program from the point where it left off. If a "run from" command is executed after the "until" breakpoint, the stack, CPU registers and memory locations may be improperly set for the code to be executed at the "run from" address.

These situations cannot be corrected within the feature set of the emulator. You must be aware of your software requirements, and the mechanism used to implement the run commands. A detailed explanation of how the run command works is given in chapter 9.

Using The Emulation Monitor

Note

This manual supports version 2.xx of the 68020 emulation software. Several important changes have been made to the emulation monitor. These changes correct defects or add new features to the monitor. Make sure that you are using the latest version of the emulation monitor in your system.

Loading the Emulation Monitor

The following rules must be followed when loading the emulation monitor:

1. The emulation monitor must be mapped to emulation RAM.

   Both program and data spaces of the monitor must be mapped to emulation RAM as opposed to ROM. The monitor transfer buffer, as well as many monitor "housekeeping" variables must be read and write accessible, and must therefore be mapped to emulation RAM.

   In addition, portions of the monitor must write to other monitor program locations for self configuration to a particular FPU coprocessor ID. Since writes to emulation ROM are always blocked, the program section, as well as the data section, of the monitor must be mapped to emulation RAM.

2. If all function codes are used, the monitor space in emulation memory must be overlaid with supervisor program space and supervisor data space.

The emulation monitor is executed in response to a level 7 interrupt. Therefore, it is always executed within supervisor space and must be located in supervisor space. If the supervisor/user function code bit is not in use, this restriction does not apply.

3. If only program and data function codes are used, the monitor space in emulation memory must be overlaid with program and data space.

The emulation software recognizes only program symbols. In the case of the monitor, the symbol addresses are assumed to be associated with the SUPR_PROG function code (since the monitor is basically an interrupt routine). Thus, when the host writes control information to, or reads information from the monitor, it must use the SUPR_PROG function code.

During monitor execution, the 68020 CPU accesses several of these same locations using the SUPR_DATA function code. This requires that the host generated SUPR_PROG, and the monitor generated SUPR_DATA accesses refer to the same memory locations. This requires that SUPR_PROG space and SUPR_DATA space be overlaid in the address range where the monitor is located. This overlay scheme uses only 1 of the 15 available mapping definitions as shown in the following example:

map fcode SUPR_PROG 0 thru 0FFFH emulation RAM Return

map_overlay fcode SUPR_DATA 0 thru 0FFFH RAM over fcode SUPR_PROG 0 Return

Resetting Into The Monitor

The "reset into monitor" facility of the emulator makes use of internal jamming circuitry to supply both an initial stack pointer and an initial program counter to the CPU. These values correspond to the values of monitor symbols SP_TEMP and RESET_ENTRY respectively. Jamming from reset appears as a sequence of eight, byte reads. The first four bytes supply the stack pointer value. The remaining four bytes provide the program counter information.

Jamming from reset occurs only if the emulator caused the reset via the "reset" softkey. If the target system asserts the CPU reset signal, the jamming circuitry is disabled and startup from reset occurs normally, with stack pointer and program counter values being supplied from memory system addresses 0-7.

The setting of the initial stack pointer value is critical to proper system operation. Since SP_TEMP is only provided as a small temporary stack for use with the monitor, the stack may be easily overflowed once a "run from ..." command is given, and the target system program begins execution. Portions of the monitor may be overwritten if the SP_TEMP stack overflows.

To ensure proper operation, be sure to either extend the SP_TEMP stack to meet target system requirements, or modify the SP_TEMP value to point to the usual target system stack. This can be done by including an appropriate "equate" statement in the monitor, while commenting out the normal SP_TEMP label in the monitor.

```
SP_TEMP     EQU    <target system stack address>
```

Another approach is to be certain that software execution as a result of the "run from ..." command properly initializes the stack pointers to values appropriate to the target system.

When the emulator is in a reset condition, one of two messages appears on the emulator status line above the softkeys. If the word "Reset" appears, the present reset condition occurred as a result of the emulator. The presence of a lower case "reset" indicates that the target system is presently asserting the CPU reset signal. The 68020 emulator can be instructed to enter the emulation monitor when a "run" command is issued after "Reset" (jamming only occurs if the reset signal is asserted by the emulator). This causes the initial program counter and initial stack pointer vector

to be ignored. Instead, the jamming circuitry supplies these values based on the current location of the monitor.

A possible difficulty exists if the target system performs some hardware and/or software initializations on reset. If "reset into monitor" is used, these initializations are not performed before monitor execution is begun.

Systems With Memory Management Units (MMU's)

MMU's present additional challenges to successfully plugging the emulator into your target system. Understanding their operational characteristics will enhance your ability to determine the best approach to your plug-in.

The MC68851 User's Manual provided by the manufacturer includes good reference material in section 2 that can be applied conceptually to nearly any MMU environment. Section 1 contains a specific overview of the MMU.

Read the user's manual before attempting to connect the emulator to a target system with an MMU.

Since emulation memory is connected to the logical bus (as opposed to the physical bus), the monitor must be resident in a logical address range that is otherwise never used.

The emulation monitor must never be swapped out of logical space and another program loaded in its place, not even temporarily. If this occurs, the host (thinking that the monitor is still present) will corrupt certain memory locations causing user program failure. The jam circuit, when reacting to a request for a monitor function, will cause a jump to an address that is not in the monitor, also resulting in user program failure.

If an operating system is present in the target system, it is normal to expect that at least part of it must remain in logical address space at all times. A good place for the monitor is to include it with the portions of the operating system that are permanently resident.

DMA is almost always present in systems with MMU's. On the physical bus, DMA is used to load programs from disc to main memory, or to perform swapping between main and secondary memory. MMU's may need to use the logical bus to perform address translation functions, or to suspend CPU operation while address translation is performed. DMA should generally be enabled while working with MMU oriented systems.

If translation tables are kept in logical memory (not usually the case), and if the tables are in emulation memory, DMA must be enabled to/from emulation memory.

Memory Access Timing Issues

Access time is the time interval during a 68020 microprocessor read cycle beginning when the 68020 microprocessor places an address on the address bus and ending when valid data is present on the microprocessor's data pins.

Appendix C contains tables listing timing comparisons between the MC68020RC12, MC68020RC16, MC68020RC20, and MC68020RC25 processors and the HP 64410C/D emulator.

For a 25 MHZ 68020 microprocessor running at maximum speed and with no wait states,

Access Time = 2.5 clocks - specification 6 - specification 27

Spec. 6 = Clock High to Address/FC/Size/RMC Valid
= 25 ns. (max),
Spec. 27 = Data-in Valid to Clock Low (Data Setup)
= 5 ns. (min),
Cycle Time = 40 ns. (min),
Clock Pulse Width = 20 ns. (min).

Therefore:

Access Time (max) = 2(40) ns. + 20 ns.–25 ns.–5 ns. = 70 ns.

For the HP 64410SC/SD 68020 emulation system, the emulator adds the following delays:

1. Address lines buffered with a 74ALS245 = 10 ns. (max)
2. Data lines buffered wiht a 74ALS245 = 10 ns. (max)
3. Cable adds a 5 ns. round trip delay

One wait state at 25 MHz

An easy way to calculate the maximum access time allowed by the emulator is to use the timing comparison tables provide in appendix C of this manual. The relevant worst case specifications for the emulator are as follows:

Spec. 6 = 39 ns. (max)
Spec. 27 = 5 ns. (min)
Cycle Time = 40 ns. (min)
Clock Pulse Width = 20 ns. (min)

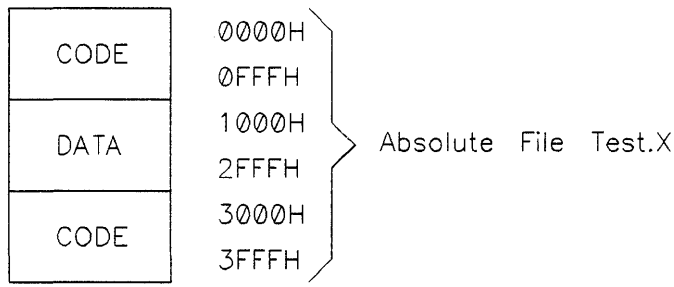
Therefore:

Access Time (max) = 3(40 ns.) + 20 ns.–39 ns.–15 ns. = 86 ns.

This is better than the 70 ns access time required by the zero-wait state target memory. However, the target system must be able to function properly with one wait state added by the emulator.

Loading An Absolute File

When an absolute file is generated, it frequently is composed of various "sections" containing code or data:



A memory map resembling that shown below might normally be generated:

Addr. Range	Attribute	Function Code
0000H - 0FFFH	EMUL RAM	SUPR_PROG
1000H - 2FFFH	EMUL RAM	SUPR_DATA
3000H - 3FFFH	EMUL RAM	USER_PROG

default = guarded

Note that upon execution of the following command, a guarded access will occur:

load memory Test.X **fcode SUPR_PROG Return**

This is due to the fact that the "load" mechanism attempts to load the entire file using the SUPR_PROG function code. In the case of Test.X (with the memory map above), address range 0000H - 0FFFH is mapped to emulation memory when the function code is SUPR_PROG. The remaining address ranges of Test.X are actually mapped to GUARDED memory when the function code is SUPR_PROG. This is because the default is set to GUARDED, and because there are no mapping definitions for SUPR_PROG covering the remaining address ranges of Test.X.

Similar symptoms would be observed with either of the following commands:

load memory Test.X **fcode SUPR_DATA**
load memory Test.X **fcode USER_PROG**

The "load memory . . ." command is defined as a vehicle to "load all memory areas" present in a given absolute file. (Guarded, as well as target and emulation memory.)

The "load memory emulation . . ." command is used to "load only areas mapped to emulation memory" in a particular absolute file.

Thus, to properly load Test.X, the following three commands would be issued:

load memory emulation Test.X **fcode SUPR_PROG**
load memory emulation Test.X **fcode SUPR_DATA**
load memory emulation Test.X **fcode USER_PROG**

The "load memory target . . ." command is provided to "load only areas mapped to target memory" in a given absolute file.

If All Else Fails . . .

If the emulator is configured properly, and the target program and monitor are loaded, unexplained behavior may still exist. This is frequently due to monitor interaction with the target software and/or hardware.

In the software category, check that it is appropriate to disable interrupts while in the monitor. Some systems with delta-time-interrupt structures for real-time clocks, operating system functions, etc., will crash if the delta-time-interrupt is not serviced within a preset time limit. The monitor can be customized to enable or disable interrupts as required. See the "Continuing Target System Interrupts While In The Emulation Monitor" section of chapter 6.

It is possible to "disable" the normal target system function of the level 7 (NMI) interrupt thru vector table modifications, and a small amount of additional monitor code.

Ensure that the program being executed is not accidentally overwriting the monitor or vice versa.

Use of the analyzer to examine software behavior is an effective means to solve emulation problems.

Obtain a listing of the monitor and the program being executed, and use the analyzer to verify proper operation of both.

Set the analyzer to trigger on the monitor entry point (MONITOR_ENTRY), with the trigger position set to the center of the trace. This will allow you to examine CPU activity surrounding the monitor entry. Your can observe the stacking activity of the level 7 interrupt, as well as emulator generated jam cycles. This will enable you to determine if the monitor is being initiated properly.

Ensure that the monitor exits and returns to the normal program properly. Set the analyzer to trigger on the monitor exit point (EXIT_MON), and observe the unstacking as a result of the RTE instruction. Be sure that the stack contents have not been corrupted, and that the program returns to the expected location.

Remember that the use of any monitor function will affect the timing of executing programs, and may be the cause of hardware and software anomalies.

Notes

6

The Emulation Monitor Program

Overview

This chapter:

- Describes the emulation monitor program.
- Provides information on modifying the monitor to use software breakpoints.
- Provides information for customizing the emulation monitor.
- Describes the emulation monitor memory requirements.
- Describes the emulation monitor linking requirements.
- Provides a flowchart of the emulation monitor program.

See chapter 5, Using the Emulator, and chapter 9, How the Emulator Works, for additional information about the emulation monitor and its interactions with the host computer and your target system.

Introduction

The emulation monitor program is the means by which many of the emulator functions are implemented. Functions implemented with the emulation monitor are:

- Read/write target memory
- Display/modify 68020 registers
- Display/modify 68881 FPU registers
- Execute user program
- Break from user program by:
  - analyzer generated break
  - keyboard break
  - software breakpoint
  - jump from user program
  - memory access violation break
- Reset into monitor
- Single step by opcode

A standard emulation monitor source file is supplied with each emulation system. This file must be assembled and linked by the user before using. Typically, the emulation monitor is assembled and then linked with the user program to form one software module. This module is then loaded into memory. The loaded monitor program enables emulation system functions to operate properly.

You can modify the emulation monitor to accommodate a particular target system or to expand the emulation monitor's capabilities. Comment delimiters must be removed from some functions in the monitor before they can function. If you modify the emulation monitor, the basic communication protocol between the emulation monitor and the emulation software must be maintained. A detailed description of the communication protocol and the standard emulation monitor is given in this chapter. A flowchart of the standard emulation monitor is also given.

The Break Function And The Emulation Monitor

The emulation break circuitry uses the NMI (INT7) resource of the processor to force the user program to be interrupted and the emulation monitor to be entered. A break can be generated for an illegal memory reference, a bus condition that the analysis card detects, a request by the emulation software, or from the keyboard.

Emulation Monitor Description

The emulation monitor is made up of the following major sections:

- the processor exception vector look-up table.
- the entry points into the monitor.
- the emulation command scanner.
- the command execution modules.

Each of these sections is discussed in the following paragraphs.

The Exception Vector Table

The emulation monitor is entered through processor exceptions. The emulation monitor program contains pseudo instructions that load the vector table with the addresses of the emulation monitor exception handlers. The emulation monitor exception table defines 68020 exception vectors for the convenience of the user.

The emulation monitor program is shipped from the factory with all of the exception vectors (except RESET and MONITOR SINGLE STEP) contained in comment fields. This is done to allow you to supply the addresses for your own exception routines. If you have not written any exception handlers, you should remove the

comment delimiters (*) from those provided in the monitor. This enables the processor to use the exception vector lookup table provided with the monitor program.

If the user application has its own RESET handler, the reset vector in the monitor must be modified to point to the user reset handler. Also, the reset-to-monitor function must be disabled. This is done by modifying the emulation configuration. You must answer no to the configuration question "Reset into the monitor?". See chapter 4 for detailed information.

Note 

The portion of the monitor defining the exception vector table is ORGed to 0H, and is not relocatable as is the rest of the monitor. When configuring the emulator, be sure to map the first block of memory (0H) to supervisor_data emulation ram. Otherwise, locate the vector table in ROM in the target system. Refer to the section in this chapter titled "LOADING THE EMULATION MONITOR" for a detailed on mapping the emulation monitor into memory.

Emulation Monitor Entry Point Routines

The emulation monitor entry point routines provide input handler routines for the various entry paths. Six separate paths are defined for monitor entry. Each path is distinguished from the others by means of a unique ENTRY_ID code which is stored upon entry into the monitor. The emulation monitor entry point routines are MONITOR_ENTRY, SPECIAL_ENTRY, SWBK_ENTRY, JSR_ENTRY, RESET_ENTRY, and EXCEPTION_ENTRY.

Monitor_entry

MONITOR_ENTRY is the entry point into the emulation monitor for breaks from the user's program. On a break to MONITOR_ENTRY, the 68020 PC and status register should be placed on the stack as is normally done when an exception oc-

curs. On entry to the monitor, the processor's registers are saved and the interrupt mask is restored (if you have modified your monitor to enable this function). The emulation monitor then executes the command scanner routines.

Special_entry

SPECIAL_ENTRY is the entry point into the monitor when the 68020 processes either a bus error or an address error exception. The only difference between this entry and MONITOR_ENTRY is that the additional words unique to the exception are taken off the stack and saved in variables.

Swbk_entry

SWBK_ENTRY is the entry point into the emulation monitor when a software breakpoint (i.e., a BKPT instruction inserted in your code by the HP 64000-UX system) is processed.

Jsr_entry

JSR_ENTRY is the entry point into the emulation monitor that should be used if the user wants to jump directly to the emulation monitor. If running in supervisor mode, you can use the instruction "JSR JSR_ENTRY" to jump to the emulation monitor. If the 68020 processor is running in user mode, a trap exception should be used. The trap vector should point to MONITOR_ENTRY.

Reset_entry

RESET_ENTRY is the entry point into the emulation monitor when the 68020 processes the reset exception. RESET_ENTRY sets up a default stack and sets the processor's registers to default values.

Exception_entry

A set of exception entry points are provided to give status messages for the ten exception vectors after reset. These exception vectors are provided for the convenience of the user and may be

deleted or modified. For more information on the exception vector entry points, see the emulation monitor source program and the section in this chapter entitled "Modifying The Exception Vector Table".

Emulation Command Scanner

The emulation command scanner normally rests in an idle loop labeled MONITOR_LOOP. The system global MONITOR_CONTROL is repetitively examined. If bit 15 is set to zero, the idle loop is resumed. If bit 15 is set to one, a command is present and the program branches to the appropriate command routine.

Bit 15 of MONITOR_CONTROL is set by the Host and cleared by the monitor program. The lower byte of MONITOR_CONTROL contains a command number against which the command table is compared. If a match is found, a command entry point will be retrieved from the table and the command will be executed. If a match is not found, the program will return to the idle loop. The command is considered complete when bit 15 of MONITOR_CONTROL is set to zero.

Emulation Command Execution Modules

The Emulation Monitor command execution modules are ARE_THERE, EXIT_MON, COPY, COPY_ALT_REG, and FIX_FPU_CODE.

Are_there

ARE_THERE is used by the host (the HP 64000-UX) to determine whether the processor is executing in the monitor or in the target system code. It can also pass an ASCII message to be displayed on the host system status line.

Exit_mon

EXIT_MON reloads the processor registers from the variables that they were stored in prior to entering the monitor. The program will then exit the monitor and return to the target system code.

Copy

COPY moves data between the monitor parameter block areas and target system memory. This command is used to modify and display target system memory.

Copy_alt_reg

COPY_ALT_REG reads from and writes to coprocessor registers.

Fix_fpu_code

FIX_FPU_CODE modifies the FPU instructions in the monitor FPU_881_COPY routine to access the proper ID_CODE, as identified in the emulation configuration session. This is necessary only for the internal FPU.

Customizing The Emulation Monitor

The emulation monitor supplied with your emulator enables all emulation features to operate in most systems. Some systems, however, require modification to the emulation monitor program in order to maximize the effectiveness of the emulator. For this reason, the source program for the monitor has been provided and is thoroughly commented. Within the code, each of the standard routines has been described to enable you to easily make your modifications.

Caution 

SYSTEM MAY BECOME UNUSABLE. Your customized portion of the emulation monitor must not exit the monitor program. Exiting the monitor will cause the entire system to become unsynchronized and, therefore, unusable.

You should not make any changes to portions of the monitor other than those described in the following paragraphs. Changes in other sections of the monitor may cause some features to stop working due to modifications on the stack, or because the information that is passed to and from the various sections has been affected.

Note 

If you have not copied the 68020 emulation monitor source program to your subdirectory, you should copy it to your subdirectory before making any modifications. To copy the emulation monitor, execute the following command:

```
cp /usr/hp64000/monitor/mon_68020.s mon_68020.s
```

You must execute the command **"chmod 666"** on the file before you modify it. It is shipped with "read-only" permissions.

You should now modify the copy in your subdirectory.

After modifying the monitor, be sure to reassemble and relink the monitor program.

Modifying The Exception Vector Table.

Find the following program block in the emulation monitor:

```
  ORG 0  ---RESET---
     DC.L SP_TEMP
     DC.L RESET_ENTRY
* ORG 8          ---BUS ERROR---
*     DC.L BE_ENTRY
* ORG $0C        ---ADDRESS ERROR---
*     DC.L AE_ENTRY
* ORG $10        ---ILLEGAL INSTRUCTION---
*     DC.L II_ENTRY
* ORG $14        ---ZERO DIVIDE---
*     DC.L ZD_ENTRY
* ORG $18        ---CHK INSTRUCTION---
*     DC.L CI_ENTRY
* ORG $1C        ---TRAPV INSTRUCTION---
*     DC.L TI_ENTRY
* ORG $20        ---PRIVILEGE VIOLATION---
*     DC.L PV_ENTRY
  ORG $24        MONITOR SINGLE-STEP ENTRY
     DC.L MONITOR_ENTRY
* ORG $24        ---TRACE---
*     DC.L T_ENTRY
* ORG $28        ---1010 EMULATOR---
*     DC.L EA_ENTRY
* ORG $2C        ---1111 EMULATOR---
*     DC.L FE_ENTRY
* ORG $34        ---CP PROTOCOL VIOLATION---
*     DC.L CPV_ENTRY
* ORG $38        --- FORMAT ERROR ---
*     DC.L FT_ENTRY
* ORG $3C        ---UNINITIALIZED INTERRUPT---
*     DC.L UI_ENTRY
* ORG $C0        ---FPCP UNORDERED CONDITION---
*     DC.L FBUC_ENTRY
* ORG $C4        ---FPCP INEXACT RESULT---
*     DC.L FIR_ENTRY
* ORG $C8        ---FPCP ZERO DIVIDE---
*     DC.L FZD_ENTRY
* ORG $CC        ---FPCP UNDERFLOW---
*     DC.L FU_ENTRY
* ORG $D0        ---FPCP OPERAND ERROR---
*     DC.L FOE_ENTRY
* ORG $D4        ---FPCP OVERFLOW---
*     DC.L FO_ENTRY
* ORG $D8        ---FPCP SIGNALING NAN---
*     DC.L FNAN_ENTRY
* ORG $E0        ---PMMU CONFIGURATION---
*     DC.L PMC_ENTRY
* ORG $E4        ---PMMU ILLEGAL OPERATION---
*     DC.L PMIO_ENTRY
* ORG $E8        ---PMMU ACCESS VIOLATION---
*     DC.L PMAV_ENTRY
```

Now, using your editor, remove the comment delimiters (*) from the start of each line of code (except the second ORG $24 statement) to make your program look as follows:

```
   ORG 0  ---RESET---
      DC.L SP_TEMP
      DC.L RESET_ENTRY
   ORG 8          ---BUS ERROR---
      DC.L BE_ENTRY
   ORG $0C        ---ADDRESS ERROR---
      DC.L AE_ENTRY
   ORG $10        ---ILLEGAL INSTRUCTION---
      DC.L II_ENTRY
   ORG $14        ---ZERO DIVIDE---
      DC.L ZD_ENTRY
   ORG $18        ---CHK INSTRUCTION---
      DC.L CI_ENTRY
   ORG $1C        ---TRAPV INSTRUCTION---
      DC.L TI_ENTRY
   ORG $20        ---PRIVILEGE VIOLATION---
      DC.L PV_ENTRY
   ORG $24         MONITOR SINGLE-STEP ENTRY
      DC.L MONITOR_ENTRY
*  ORG $24        ---TRACE---
*     DC.L T_ENTRY
   ORG $28        ---1010 EMULATOR---
      DC.L EA_ENTRY
   ORG $2C        ---1111 EMULATOR---
      DC.L FE_ENTRY
   ORG $34        ---CP PROTOCOL VIOLATION---
      DC.L CPV_ENTRY
   ORG $38        --- FORMAT ERROR ---
      DC.L FT_ENTRY
   ORG $3C        ---UNINITIALIZED INTERRUPT---
      DC.L UI_ENTRY
   ORG $C0        ---FPCP UNORDERED CONDITION---
      DC.L FBUC_ENTRY
   ORG $C4        ---FPCP INEXACT RESULT---
      DC.L FIR_ENTRY
   ORG $C8        ---FPCP ZERO DIVIDE---
      DC.L FZD_ENTRY
   ORG $CC        ---FPCP UNDERFLOW---
      DC.L FU_ENTRY
   ORG $D0        ---FPCP OPERAND ERROR---
      DC.L FOE_ENTRY
   ORG $D4        ---FPCP OVERFLOW---
      DC.L FO_ENTRY
   ORG $D8        ---FPCP SIGNALING NAN---
      DC.L FNAN_ENTRY
   ORG $E0        ---PMMU CONFIGURATION---
      DC.L PMC_ENTRY
   ORG $E4        ---PMMU ILLEGAL OPERATION---
      DC.L PMIO_ENTRY
   ORG $E8        ---PMMU ACCESS VIOLATION---
      DC.L PMAV_ENTRY
```

End out of your edit session, making sure that you save your changes.

By removing the comment delimiters from this section of the monitor, you have made the exception vector table usable. The table provides all addresses that the monitor needs to operate.

Continuing Target System Interrupts While In The Emulation Monitor

The processor interrupt mask can be restored to its prebreak value to enable target system interrupts while in the monitor. You must edit the monitor program if you want to enable the interrupts while running in the emulation monitor.

Under the JUMP_ENTRY label, you will find a commented section that describes re-enabling the interrupts.

```
*********************************************************************
*
*  IN ORDER TO KEEP USER INTERRUPTS ENABLED THE FOLLOWING SEGMENT
*  WILL RESTORE THE INTERRUPT MASK TO ITS PRE-BREAK STATE.
*
*  USER INTERRUPT ROUTINES ARE EXPECTED TO PRESERVE ALL REGISTERS
*  IF INTERRUPTS ARE TO BE ENABLED WHILE IN THE EMULATION MONITOR.
*
*********************************************************************

    BRA     SKIP_INT_ENABLE    ; DEFAULT IS NOT RE-ENABLE INTERRUPTS
    MOVE.W  PSTATUS,D0         ; GET COPY OF PRE-BREAK STATUS REG.
    ORI.W   #$0F8FF,D0         ; COVER ALL BITS EXCEPT INTERRUPT
    MOVE.W  SR,D1          ;  MASK.
    AND.W   D1,D0
    MOVE.W  D0,SR          ; RESTORE INTERRUPT MASK.
```

Comment the instruction "BRA SKIP_INT_ENABLE" to use the interrupts while in the monitor. Be sure to save your changes.

Sending Messages From the User Program To the Emulator Display

The PUT_MONITOR_MSG routine in the emulation monitor gives you a way to send messages to the display status line. In order to use this feature, you must do the following steps:

1. Define the message in your user code.
2. Set a trap vector to point to the PUT_MONITOR_MSG routine.
3. Initiate the appropriate trap. This will cause a "message breakpoint" and leave the processor running in the emulation monitor.
4. If you want to continue execution of your user program, your program should pop one long word off the stack to clean up the stack after the trap.

An example program implementing the "message breakpoint" is shown in figure 6-1.

```
        CHIP  68020

*************************************************************************
* The following program segment demonstrates how to pass a monitor      *
* message to the emulator. The user program should jump to FINISH       *
* when it has completed, and the message "End of program " will then    *
* appear on the emulation STATUS line.                                  *
*************************************************************************
        XDEF    FINISH            ; global declaration

        XREF    PUT_MONITOR_MSG   ; external declaration
*************************************************************************
* Note that the TRAP vector for TRAP #0 ($80) has been set up in this *
* example program. If this program were to reside in user space, or *
* were not linked to the emulation monitor, you would want to put the *
* TRAP vector in the emulation monitor.                                 *
***********(************************************************************

        ORG     $80
        DC.L    PUT_MONITOR_MSG  ; TRAP #0 vector

*************************************************************************
* The following segments are relocatable.        *
*************************************************************************

        SECT    PROG

FINISH  PEA     MESSAGE          ; push addr of message onto the stack
        TRAP    #0               ; trap #0 sends the monitor message,
                                 ;  and leaves us running in the monitor.

        TST.L   (SP)+            ; clean up the stack (pop a long word)
LOOP    BRA     LOOP

*************************************************************************
* Messages MUST be in supervisor/user data space (if function codes  *
* are meaningful in your application); usually, this means being in  *
* a different section than the code is in.  *
*************************************************************************

        SECT  DATA

MESSAGE DC.B  MSG_END-MESSAGE-1 ; size of message (up to 30 chars)
        DC.B  'End of program ' ; body of message
MSG_END DC.B  0                 ; NULL terminator

END
```

Figure 6-1. Monitor Message Routine

Emulation Monitor Memory Requirements For The 68020

The memory available in the emulation hardware is divided into 256-byte blocks of address space by the emulation system. Each 256-byte block begins on an even address multiple of 100H.

The relocatable program area of the emulation monitor requires approximately 4200 bytes of memory. You can determine this if you look at the **MODULE SUMMARY** section of the linker listing file (figure 6-2). You can see, in this example, that the emulation monitor begins at address 20000H and ends at address 21071H. The program takes up 0AF8 hexadecimal locations of memory. The value 1071H is approximately 4200 decimal. Therefore, the emulation monitor can be mapped into 17 256-byte blocks of memory.

These memory requirements assume that the blocks each start on a 256-byte boundary and that the standard emulation monitor is being loaded. To check the memory requirements for the emulation monitor being used, the linker listing file should be checked.

```
MODULE SUMMARY
--------------

MODULE          SECTION:START        SECTION:END       FILE

.
.
.
monitor                :00000000            :0000002F  /usr/hp64000/env/hp64410
                                                                    /mon_68020.o
                       :000000C0            :000000DB
                mon_prog:00020000    mon_prog:0002087D
                mon_data:00020880    mon_data:00021071
.
.
.
```

Figure 6-2. Example Mon_68020 Listing File

The monitor program must reside in supervisor space in emulation RAM. The monitor must be mapped as overlaid supervisor program and supervisor data spaces. See the section "Loading The Emulation Monitor" in this chapter for details.

Linking The Emulation Monitor

The emulation monitor must be assembled and linked before it can be used by the emulation system. It can be linked with the target system code to produce one absolute file or it can be linked by itself.

It is possible, using the options to the "load" command, to load the monitor into emulation memory, then load your target system code into the target system. You first "load memory emulation <FILE>" and then "load memory target <FILE>".

Loading The Emulation Monitor

The following rules must be followed when loading the emulation monitor:

1. The emulation monitor must be mapped to emulation RAM.

   Both program and data spaces of the monitor must be mapped to emulation RAM as opposed to ROM. The monitor transfer buffer, as well as many monitor "housekeeping" variables must be read and write accessible, and must therefore be mapped to emulation RAM.

In addition, portions of the monitor must write to other monitor program locations for self configuration to a particular FPU coprocessor ID. Since writes to emulation ROM are always blocked, the program section, as well as the data section, of the monitor must be mapped to emulation RAM.

2. If all function codes are used, the monitor space in emulation memory must be overlaid with supervisor program space and supervisor data space.

   The emulation monitor is executed in response to a level 7 interrupt. Therefore, it is always executed within supervisor space and must be located in supervisor space. If the supervisor/user function code bit is not in use, this restriction does not apply.

3. If only program and data function codes are used, the monitor space in emulation memory must be overlaid with program and data space.

The emulation software recognizes only program symbols. In the case of the monitor, the symbol addresses are assumed to be associated with the SUPR_PROG function code (since the monitor is basically an interrupt routine). Thus, when the host writes control information to, or reads information from the monitor, it must use the SUPR_PROG function code.

During monitor execution, the 68020 CPU accesses several of these same locations using the SUPR_DATA function code. This requires that the host generated SUPR_PROG, and the monitor generated SUPR_DATA accesses refer to the same memory locations. This requires that SUPR_PROG space and SUPR_DATA space be overlaid in the address range where the monitor is located. This overlay scheme uses only 1 of the 15 available mapping definitions as shown in the following example:

```
map fcode SUPR_PROG 0 thru 0FFFH emulation RAM
map_overlay fcode SUPR_DATA 0 thru 0FFFH RAM over
    fcode SUPR_PROG 0
```

Emulation Monitor Flowchart

The emulation monitor flowchart is given in figure 6-3.

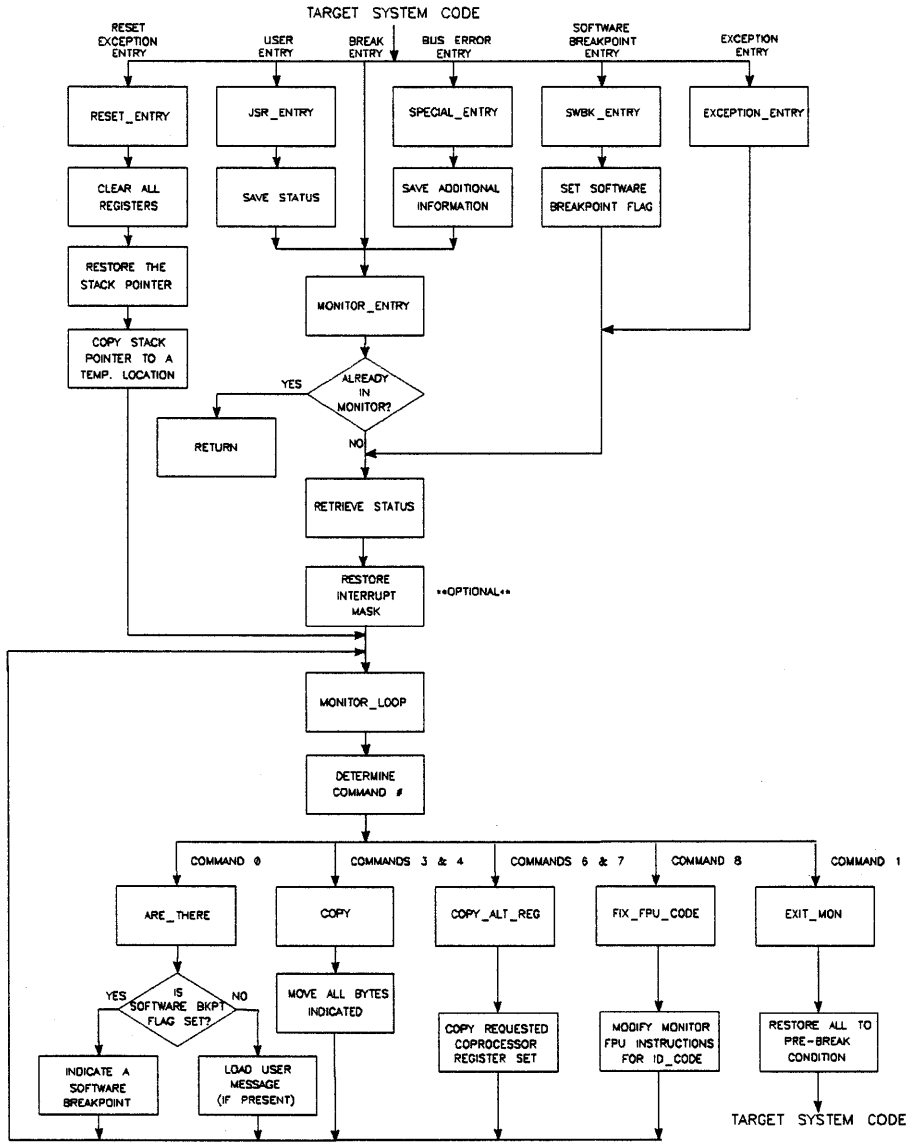


Figure 6-3. Emulation Monitor Flowchart

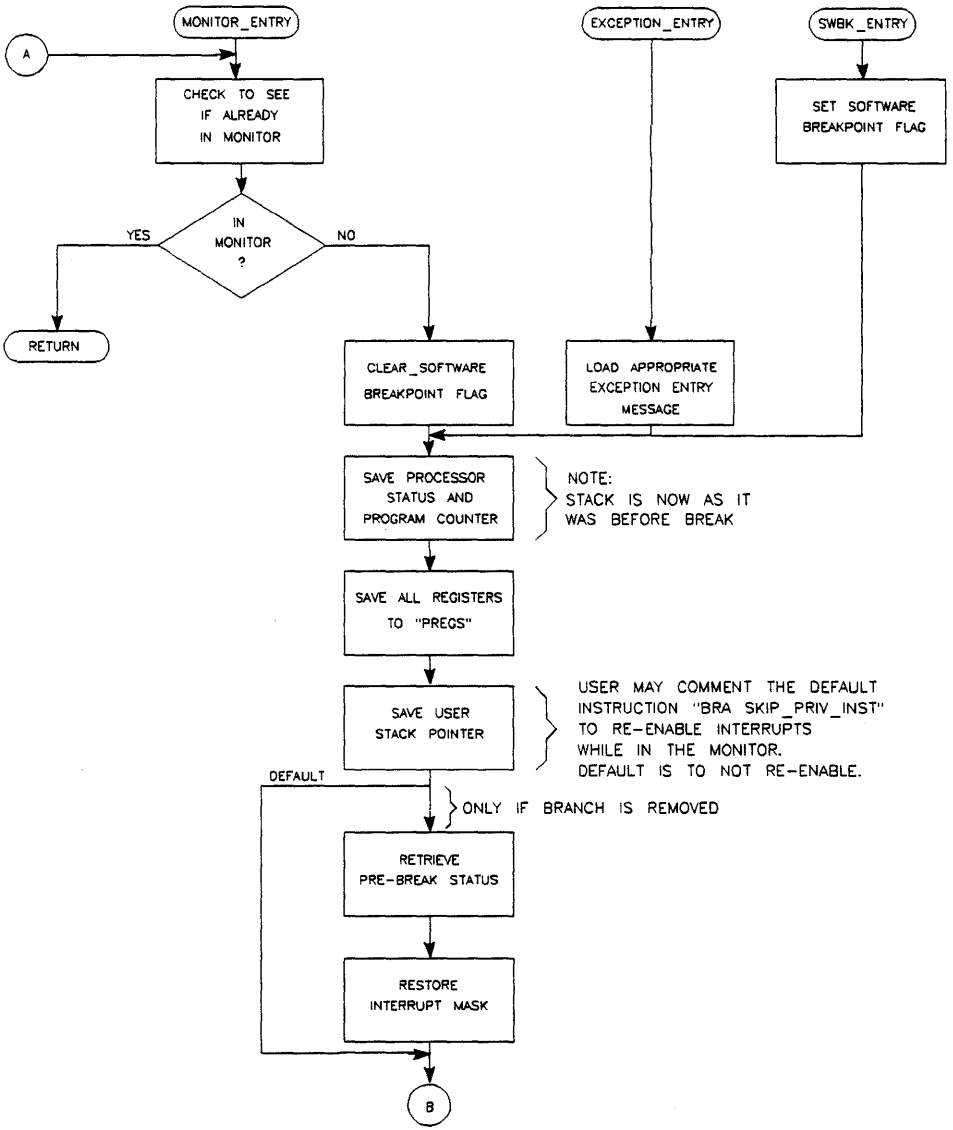


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

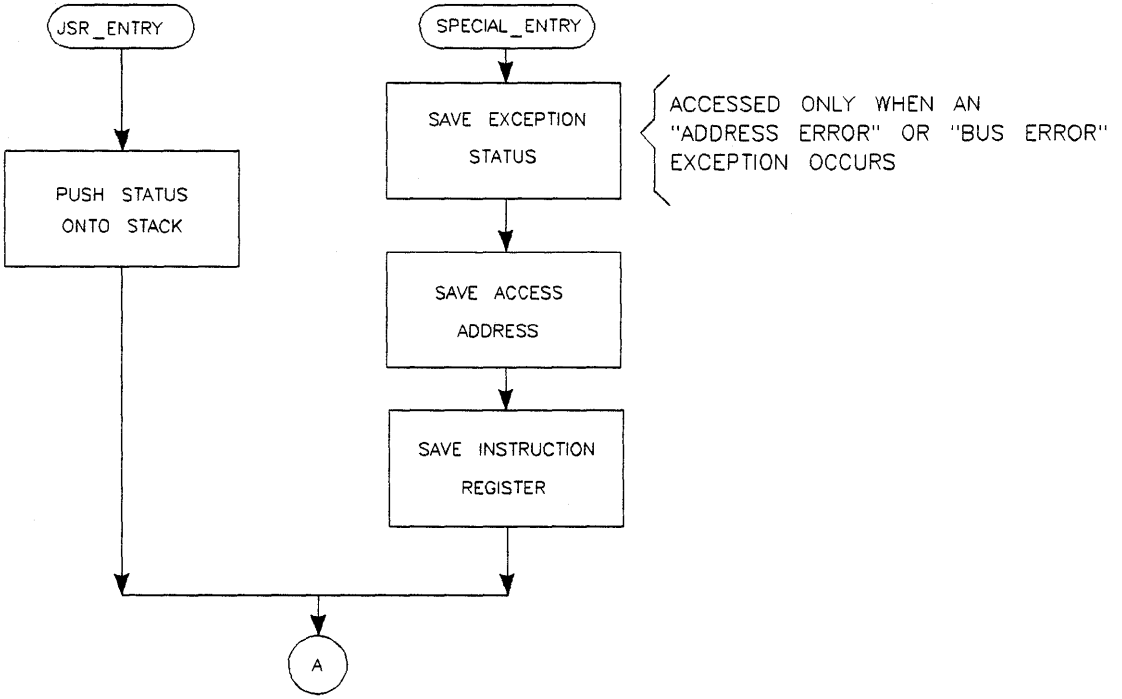


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

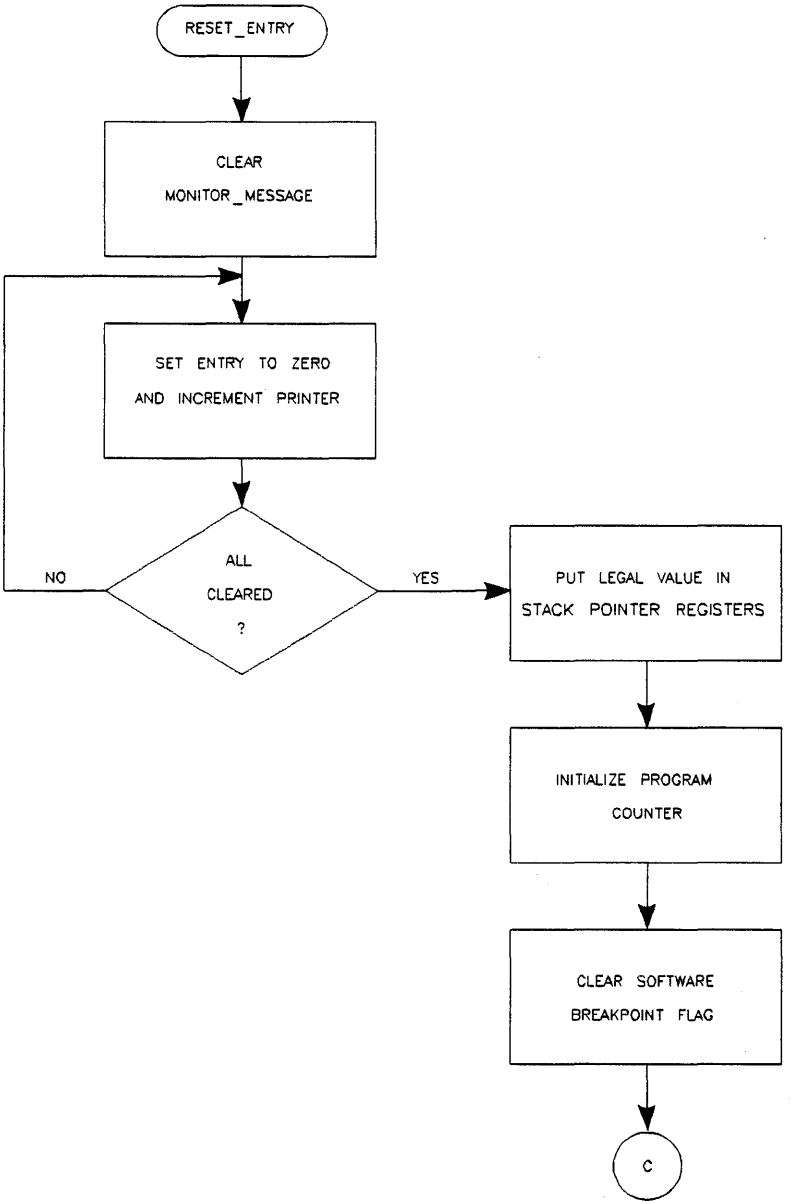


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

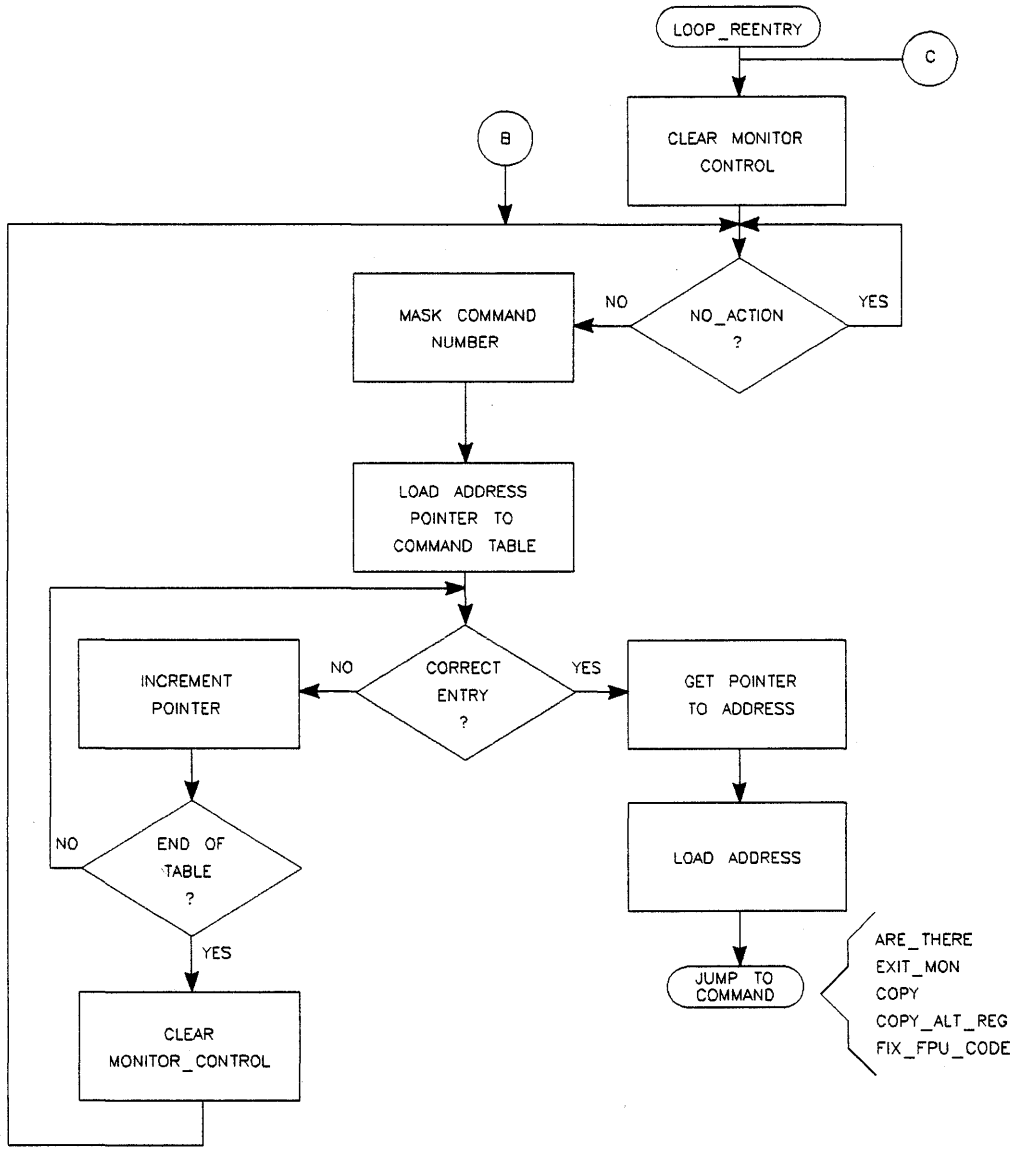


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

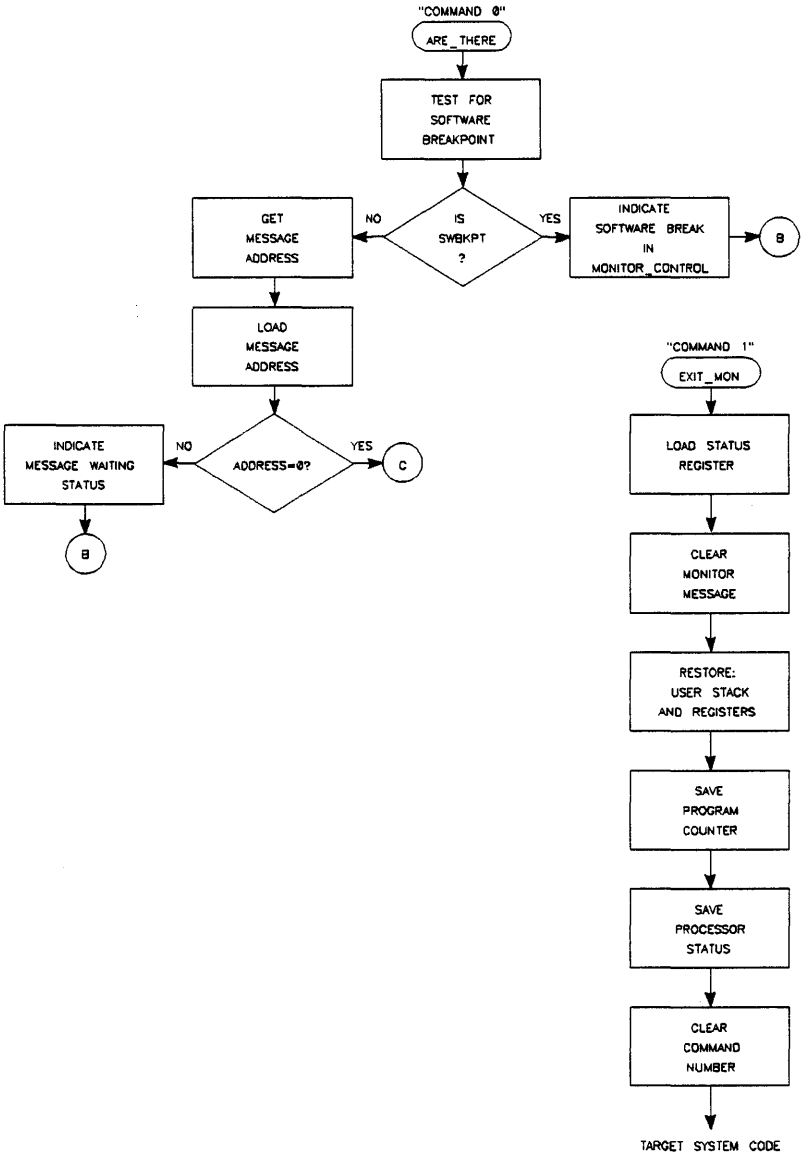


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

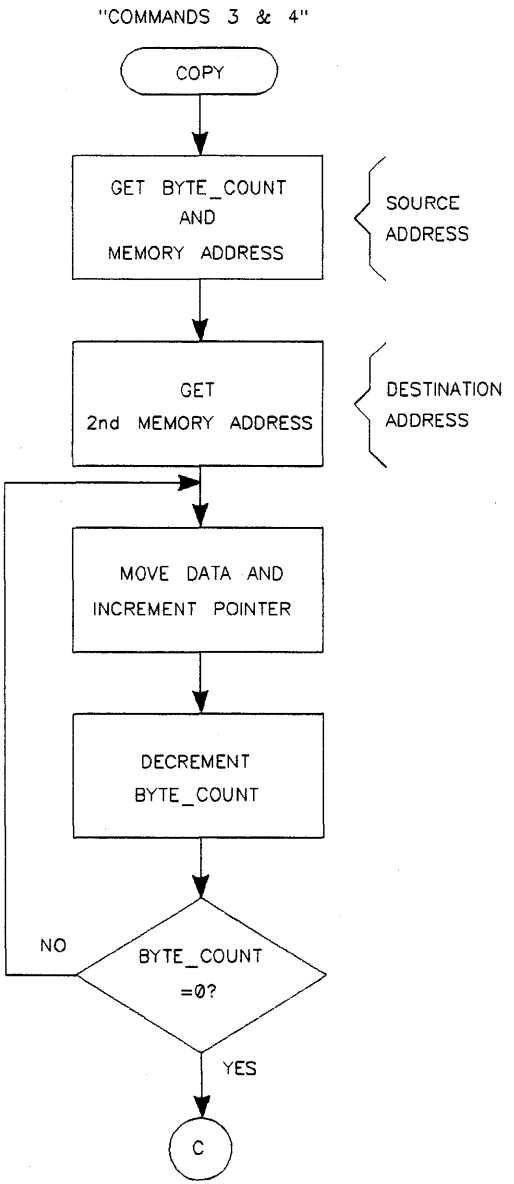


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

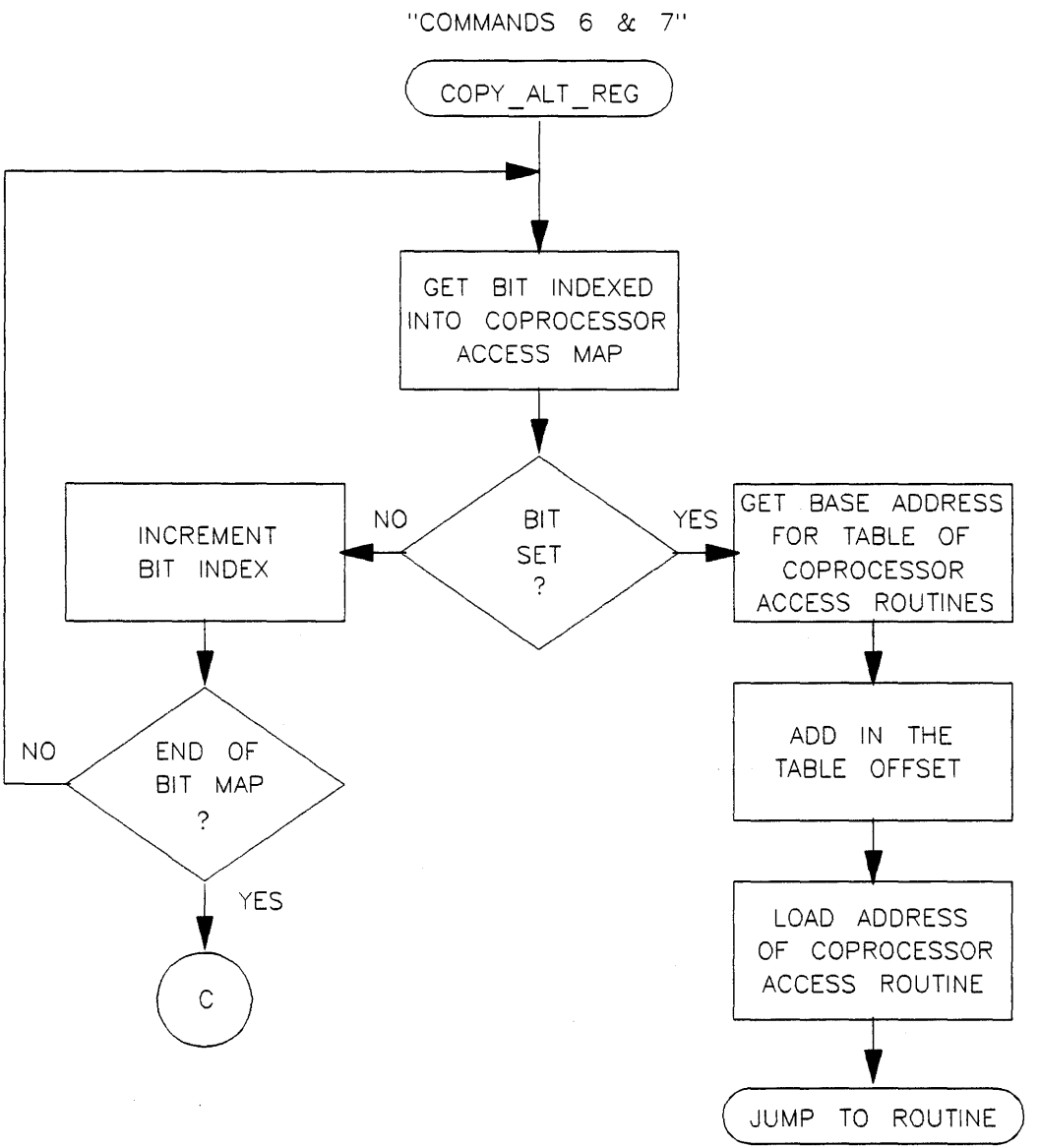


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

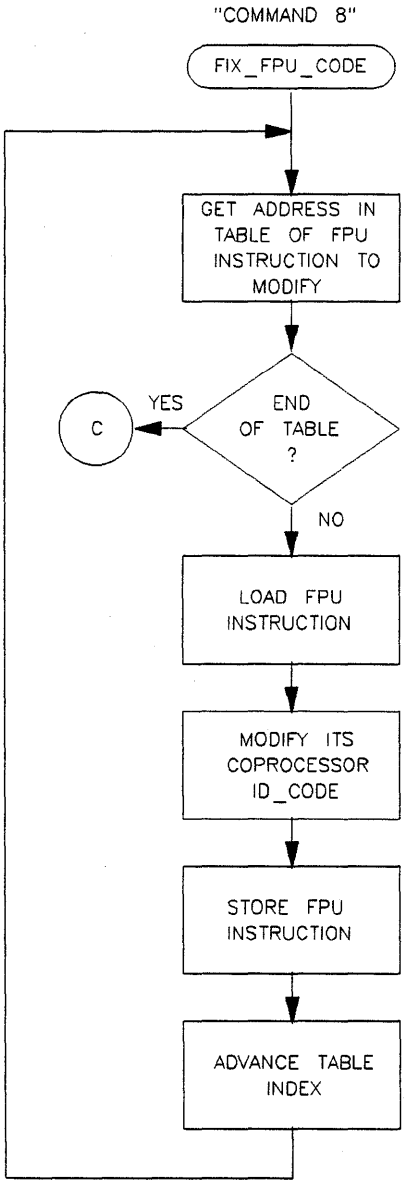


Figure 6-3. Emulation Monitor Flowchart (Cont'd)

7

Using Custom Coprocessors

Overview

This chapter provides the following information:

- A discussion of the requirements for using custom coprocessors.
- A detailed description of the custom coprocessor format file.
- A detailed description of how to modify the emulation monitor for use with custom coprocessors.
- A description of the emulation configuration questions related to custom coprocessors.

Introduction

The 68020 emulator has the capability to access floating point coprocessors, memory management units, and other coprocessors in your target system. You can both display and modify coprocessor register sets. A floating point coprocessor (FPU) is provided in the 68020 emulation pod.

In order to use custom coprocessors with the emulator, you must:

- provide a custom register format file defining the coprocessor address, size, and name and defining the register display format.
- modify the emulation monitor program to include a storage buffer for the coprocessor registers, read/write routines to access coprocessor registers, and a pointer to the coprocessor read/write routines.
- specify the custom register format file to the emulator during emulation configuration.

A default custom register format file is provided with your emulation software for use with the emulator's internal FPU. Read/write routines for the internal FPU are provided in the emulation monitor program. If you are using only the internal FPU coprocessor, answer "yes" to the emulation configuration question "Enable Internal 68881 FPU?" and select the default custom register file. This file is named **/usr/hp64000/inst/emul32/0400/0001/custom_spec**.

When the internal FPU is used along with other coprocessors, the pointer in the emulation monitor to the internal FPU read/write routines is set up programmatically by the emulation software.

The Custom Register Format File

A custom register format file must specify the internal FPU (if used) and any other coprocessor you want to use with emulation. This file specifies:

- which coprocessors should be used,
- which coprocessor space the coprocessors should be located in,
- how large the register buffer should be for transfers,
- what the display should look like for each coprocessor,
- and what register names there are for register modifies.

This file is read when the emulation configuration file is processed. The custom register format specification is fairly simple. For each coprocessor register set defined in the file, the following items must appear in the order specified:

1. the coprocessor address
2. the coprocessor size
3. the coprocessor name
4. the display spec

You may place comments in C language format (enclosed by "/*" and "*/") or blank lines before or after any register set, as well as between the specification fields. You can specify C language format include files using a control line of the form:

```
#include "filename"
    or
#include <filename>
```

where the register set description could be placed in the include file. Filename must be the full pathname for the include file.

Using include files simplifies your custom register specification file and allows you to easily remove a register set from the specification file, if necessary.

A listing of a sample custom register specification file is shown in figure 7-1 at the end of this section. Figures 7-2 and 7-3 shows how

the same file could be written using a sample include file and include command lines.

Address specification

The address specification is of the form:

ADDR=n

where n is the coprocessor identification code that defines the coprocessor space. The address must be a number between 0 and 7, inclusive. If two register sets in the format file have the same address, only the last specified register set is used. The first register set is ignored. ADDR 0 is reserved for an MMU, if present. The address specified for the "fpu" coprocessor must match the internal FPU coprocessor identification code if the internal FPU is used.

Size Specification

The size specification is of the form:

SIZE=n

where n is the size (in bytes) of the register set transfer buffer. The transfer buffer is used to transfer the register contents between the emulation monitor and the host system. This number must be between 0 and 1020, inclusive.

Name Specification

The coprocessor name specification is of the form:

NAME="string"

where string is a unique name for the coprocessor. If the name is not unique, any previous register specs with the same name will be ignored. The string must only contain alphanumeric characters. Register set names are available on softkeys during display, copy, and modify commands. Register set names are also placed in the header of the register display if the coprocessor set is active during the display. The name "fpu" is reserved for the internal FPU, if used.

Register Set Display Specification

The register set display specification is enclosed by two lines as follows:

```
DISPLAY_START
    <display specification>
DISPLAY_END
```

The DISPLAY_START and the DISPLAY_END lines cannot have any trailing blanks. Any statements within these lines are used to generate the register display. These lines also provide information to the emulator for setting up register names for the modify command. Register specifications have the form:

NAME %OFFSET.WIDTH

where NAME is the name that the register should be referenced by during display and modify commands.

OFFSET is the index into the register buffer (in bytes) to the location of the register contents.

WIDTH is the register width (in bytes).

All other text and white space in the register specification is presented in the display exactly as specified in the format file.

Using the Internal FPU

The internal FPU is a special case of the coprocessor registers. There must be an entry for it in the coprocessor register format file if the internal FPU is enabled. This entry must have the coprocessor name "fpu" and the address must match the coprocessor id specified for the internal FPU during configuration.

```
            /***********************************************/
            /*                                             */
            /*   COPROCESSOR DISPLAY FORMAT SPECIFICATIONS  */
            /*                                             */
            /***********************************************/
/* This file contains the display format specifications for all coprocessors */
/* configured for this system.  It will always contain the display spec for */
/* the internal  68881 fpu,  but may also contain up to 7 other  coprocessor */
/* specifications.                                                           */
/*                                                                           */
/* The entry below  describes the format for the 68881 fpu,  and may be used */
/* as an example.  There are several  pieces of data which MUST be  supplied */
/* for each specification:                                                   */
/*                                                                           */
/*   ADDR=n,  where n is in the range 0-7.   This is the coprocessor id-code */
/*            for the current entry. Please note that ADDR=0 is reserved for */
/*            an MMU (if  present),  and that all ADDR  designations  should */
/*            appear only once in this file.                                 */
/*                                                                           */
/*   SIZE=n,  where  0 < n < 1020 bytes.  SIZE describes the number of bytes */
/*            in the monitor  register  buffer the user has defined for this */
/*            coprocessor.                                                   */
/*                                                                           */
/*   NAME="string",   where  "string"  is the UNIQUE  name  of  the  current */
/*            coprocessor.   The name is made up of alphanumeric characters  */
/*            only.   This  name  will  show  up  on a  softkey  when        */
/*            attempting to display/modify registers within emulation.       */
/*                                                                           */
/*   DISPLAY_START   marks the  start  of the  display  format  spec for the */
/*            current coprocessor.                                           */
/*                                                                           */
/*   DISPLAY_END     marks  the end of the  display  spec,  and also the end */
/*            of the information for the current coprocessor.   A new speci- */
/*            fication may follow each DISPLAY_END.                          */
/*                                                                           */
/* Within the bounds of  DISPLAY_START  and  DISPLAY_END  is the information */
/* needed to generate  the  display  for each  coprocessor.   Each  register */
/* description contains a name field and a register format field. The format */
/* field is in the form:                                                     */
/*                                                                           */
/*     %OFFSET.WIDTHr,  where OFFSET is the index into the  register  buffer */
/*            defined in the monitor (in bytes),  and  WIDTH is the width of */
/*            the  register (also in bytes).   All other text,  white space, */
/*            etc, are preserved in the display.                             */
```

Figure 7-1. Sample Custom Register Specification File

```
/************************************/
/*                                  */
/* INTERNAL 68881 FPU SPECIFICATION */
/*                                  */
/************************************/

ADDR=1        /* the fpu id-code (special: set by configuration) */
SIZE=108      /* number of bytes in the fpu register buffer */
NAME="fpu"    /* name of the fpu coprocessor (do not change) */

DISPLAY_START
    FP0 %00.12r  FP1 %12.12r  FPCR %96.4r
    FP2 %24.12r  FP3 %36.12r  FPSR %100.4r
    FP4 %48.12r  FP5 %60.12r FPIAR %104.4r
    FP6 %72.12r  FP7 %84.12r

DISPLAY_END

/* Other custom coprocessor display formats follow... */
```

Figure 7-1. Sample Custom Register Spec. File (Cont'd)

```
/************************************/
/*                                  */
/* INTERNAL 68881 FPU SPECIFICATION */
/*                                  */
/************************************/
ADDR=1        /* the fpu id-code (special: set by configuration) */
SIZE=108      /* number of bytes in the fpu register buffer */
NAME="fpu"    /* name of the fpu coprocessor (do not change) */

DISPLAY_START
    FP0 %00.12r  FP1 %12.12r  FPCR %96.4r
    FP2 %24.12r  FP3 %36.12r  FPSR %100.4r
    FP4 %48.12r  FP5 %60.12r FPIAR %104.4r
    FP6 %72.12r  FP7 %84.12r

DISPLAY_END
```

Figure 7-2. Custom Reg. Spec. Include File fpu_spec

```
            /*********************************************/
            /*   COPROCESSOR DISPLAY FORMAT SPECIFICATIONS   */
            /*                                             */
            /*********************************************/
/* This file contains the display format specifications for all coprocessors */
/* configured for this system.   It will always contain the display spec for */
/* the internal  68881 fpu,  but may also contain up to 7 other  coprocessor */
/* specifications.                                                           */
/*                                                                           */
/* The entry below  describes the format for the 68881 fpu,  and may be used */
/* as an example.  There are several  pieces of data which MUST be  supplied */
/* for each specification:                                                   */
/*                                                                           */
/*   ADDR=n,  where n is in the range 0-7.   This is the coprocessor id-code */
/*            for the current entry. Please note that ADDR=0 is reserved for */
/*            an MMU (if  present),  and that all ADDR  designations  should */
/*            appear only once in this file.                                 */
/*                                                                           */
/*   SIZE=n,  where  0 < n < 1020 bytes.  SIZE describes the number of bytes */
/*            in the monitor  register  buffer the user has defined for this */
/*            coprocessor.                                                   */
/*                                                                           */
/*   NAME="string",   where  "string"  is the UNIQUE  name  of  the  current */
/*            coprocessor.   The name is made up of alphanumeric characters  */
/*            only.   This  name  will  show  up  on a  softkey  when        */
/*            attempting to display/modify registers within emulation.       */
/*                                                                           */
/*   DISPLAY_START   marks the  start  of the  display  format  spec for the */
/*            current coprocessor.                                           */
/*                                                                           */
/*   DISPLAY_END     marks  the end of the  display  spec,  and also the end */
/*            of the information for the current coprocessor.   A new speci- */
/*            fication may follow each DISPLAY_END.                          */
/*                                                                           */
/*                                                                           */
/* Within the bounds of  DISPLAY_START  and  DISPLAY_END  is the information */
/* needed to generate  the  display  for each  coprocessor.   Each  register */
/* description contains a name field and a register format field. The format */
/* field is in the form:                                                     */
/*                                                                           */
/*     %OFFSET.WIDTHr,  where OFFSET is the index into the  register  buffer */
/*            defined in the monitor (in bytes),  and  WIDTH is the width of */
/*            the  register (also in bytes).   All other text,  white space, */
/*            etc, are preserved in the display.                             */

#include "/users/em68020/custom_spec/fpu_spec"
#include "/users/em68020/custom_spec/mmu_spec"
    .
    .
    .
```

Figure 7-3. Custom Reg. Spec. File Using Include Files

Emulation Monitor Changes

In order to access coprocessor register sets, you must make some minor changes to the emulation monitor. You must declare a register buffer for storing the coprocessor register values, modify two table entries, and provide register buffer read/write routines for each coprocessor register set that the emulation monitor will access.

Defining a Coprocessor Register Buffer

A coprocessor register buffer must be allocated in the emulation monitor for each custom coprocessor you use with the emulator. The emulator uses this buffer for storing register values read from or written to the custom coprocessor. A buffer (FPU_881_REGS) for the internal FPU is provided with the emulation monitor program. This buffer is declared in the emulation monitor as follows:

```
FPU_881_REGS
FP_0_7          DS.L      24
FPCRT           DC.L      0
FPSRT           DC.L      0
FPIART          DC.L      0
FPU_881_END
```

Locate this declaration in the emulation monitor program and insert your custom coprocessor register buffer declarations immediately following it in the emulation monitor. For example, if you are using an MC68851 Memory Management Unit in your target system, you might add the following register buffer declaration:

```
MMU_851_REGS
MMUCPR          DS.L      2
MMUDMA          DS.L      2
MMUSRP          DS.L      2
MMUTC           DC.L      0
MMUPCSR         DC.W      0
MMUPSR          DC.W      0
MMUCAL          DC.B      0
MMUVAL          DC.B      0
MMUSCC          DC.B      0
MMUAC           DC.W      0
MMUBD0_7        DC.L      4
MMUBC0_7        DC.L      4
MMU_851_END
```

Modifying The MON_ALT_BUFFER Table

After declaring your register buffers, you need to modify the MON_ALT_BUFFER table. This table has entries labeled "COPROC_REG_n", where n is the coprocessor identification number. The coprocessor identification numbers specified in the format file must have their corresponding table entry set to point to a buffer that will be used to transfer the register data to and from the monitor. These are the buffers that you declared in the previous section. The default MON_ALT_BUFFER table provided with your emulation monitor is shown in the following listing:

```
MON_ALT_BUFFER

COPROC_REG_0    DC.L        0
COPROC_REG_1    DC.L        0
COPROC_REG_2    DC.L        0
COPROC_REG_3    DC.L        0
COPROC_REG_4    DC.L        0
COPROC_REG_5    DC.L        0
COPROC_REG_6    DC.L        0
COPROC_REG_7    DC.L        0
```

For example, if you want to an MMU in your target system, you might want to modify the the MON_ALT_BUFFER table as follows:

```
MON_ALT_BUFFER

COPROC_REG_0    DC.L        MMU_851_REGS
COPROC_REG_1    DC.L        0
COPROC_REG_2    DC.L        0
COPROC_REG_3    DC.L        0
COPROC_REG_4    DC.L        0
COPROC_REG_5    DC.L        0
COPROC_REG_6    DC.L        0
COPROC_REG_7    DC.L        0
```

Note that the MMU coprocessor should be assigned identification number 0.

Modifying The MON_ALT_REGISTERS Table

The second table you must change is under the symbol "MON_ALT_REGISTERS". This table has entries labeled "COPROC_LOAD_n", where n is the coprocessor identification number. These entries point to a coprocessor's read/write routine. A read/write routine (FPU_881_COPY) is provided in the emulation monitor for use with the internal FPU. The default MON_ALT_REGISTERS table provided with your emulation monitor is shown in the following listing:

```
MON_ALT_REGISTERS

COPROC_LOAD_0    DC.L       0
COPROC_LOAD_1    DC.L       0
COPROC_LOAD_2    DC.L       0
COPROC_LOAD_3    DC.L       0
COPROC_LOAD_4    DC.L       0
COPROC_LOAD_5    DC.L       0
COPROC_LOAD_6    DC.L       0
COPROC_LOAD_7    DC.L       0
```

If you want to use an MMU in your target system as in the previous example, you would modify the the MON_ALT_BUFFER table as follows:

```
MON_ALT_REGISTERS

COPROC_LOAD_0    DC.L       MMU_851_COPY
COPROC_LOAD_1    DC.L       0
COPROC_LOAD_2    DC.L       0
COPROC_LOAD_3    DC.L       0
COPROC_LOAD_4    DC.L       0
COPROC_LOAD_5    DC.L       0
COPROC_LOAD_6    DC.L       0
COPROC_LOAD_7    DC.L       0
```

where MMU_851_COPY is the copy routine you have written for your MMU registers.

Writing Coprocessor Copy Routines

The coprocessor copy routine must both read from and write to the coprocessor registers. If the emulation monitor symbol "MON_COMMAND" contains the value "6", then the routine should perform a read into the register data buffer specified above. If the symbol = 7, the routine should write the register set using the values in the register data buffer.

The internal FPU read/write routine (FPU_881_COPY) is shown in the following listing. For the example given in the pre-

vious section, you would need to write a copy routine for your MMU labeled **MMU_851_COPY**. The internal FPU copy routine provides you with a good example of how to write your copy routine.

```
*****************************************************************
*
*    FPU_881_COPY IS A SYSTEM GLOBAL ROUTINE THAT TRANSFERS THE FPU
*    REGISTERS TO/FROM THE FPU_881_REGS DATA AREA. IT IS ALWAYS
*    PRESENT SINCE THE INTERNAL 881 FPU REQUIRES IT.
*
*    FPU_881_COPY MAY BE USED AS AN EXAMPLE LOAD/UNLOAD ROUTINE FOR
*    OTHER COPROCESSORS.
*
*****************************************************************

FPU_881_COPY
*
*    FOR COPROCESSOR LOAD/UNLOAD ROUTINES THE VARIABLE MON_COMMAND
*    WILL CONTAIN A '6' TO INDICATE A READ_REGS COMMAND, OR A '7'
*    FOR A WRITE_REGS COMMAND.
*
CMPI.L    #6,MON_COMMAND        ; IS THIS A READ_ALT_REGS COMMAND?
BEQ       FPU_881_READ          ; YES, GO DO THE READ

FPU_881_WRITE
*
*    LOCAL COPY OF FPU DATA --> FPU
*
LEA       FPU_881_REGS,A0            ; GET PTR TO THE FPU REG DATA
FP_A FSAVE     -(SP)                      ; SAVE FPU CONTEXT
FP_B FMOVEM.X  (A0)+,FP0-FP7              ; WRITE OUT THE FPU DATA REGS
FP_C FMOVEM.L  (A0)+,FPCR/FPSR/FPIAR      ; WRITE OUT THE FPU CONTROL REGS
FP_D FRESTORE  (SP)+                      ; RESTORE FPU CONTEXT
JMP       LOOP_REENTRY               ; RETURN TO MONITOR CMD LOOP

FPU_881_READ
*
*    FPU --> LOCAL COPY OF FPU DATA
*    LEA       FPU_881_END,A0             ; GET PTR TO END OF FPU DATA
FP_E FSAVE     -(SP)                      ; SAVE FPU CONTEXT
FP_F FMOVEM.L  FPCR/FPSR/FPIAR,-(A0)      ; READ THE FPU CONTROL REGS
FP_G FMOVEM.X  FP0-FP7,-(A0)              ; READ THE FPU DATA REGISTERS
FP_H FRESTORE  (SP)+                      ; RESTORE FPU CONTEXT
JMP       LOOP_REENTRY               ; RETURN TO MONITOR CMD LOOP

*****************************************************************
*
*    CUSTOM COPROCESSOR REGISTER LOAD/UNLOAD ROUTINES (IF ANY) SHOULD
*    BE INSERTED INTO THE MONITOR HERE. PLEASE NOTE THAT THE DEFAULT
*    COPROCESSOR ID FOR THE ASSEMBLER IS 1. IN ORDER FOR THE ASSEMBLER
*    TO GENERATE THE CORRECT CODE FOR OTHER IDs, THE ASSEMBLER FLAG
*    "FOPT ID=n", n=1-7, SHOULD BE USED APPROPRIATELY.
*
*****************************************************************
```

Answering Emulation Coprocessor Configuration Questions

After modifying the emulation monitor, you must reassemble the emulation monitor and relink your emulation monitor with your user file.

The final step in setting up your 68020 emulator to use custom coprocessors is to answer the emulation configuration questions relating to custom coprocessors. In the default emulation configuration, you will be asked the question:

Enable internal 68881 FPU?

If you answer **yes**, you can use both the internal FPU and other coprocessors during the emulation session. The emulator will prompt you for the coprocessor identification number you want to use for the FPU.

If you are using only the internal FPU, you do not need to make any modifications to the default custom register file or to the emulation monitor.

If you answer **no**, the next question the emulator asks is:

Any custom coprocessors?

Answer **yes** to enable use of custom coprocessors.

If you answered "yes" to either of the above questions, the next question will be:

Name of custom register format file?

Enter the full pathname of your custom register format file.

Complete the remainder of the emulation configuration questions and save your changes to a configuration file. You are now ready to run emulation using custom coprocessors.

A complete and detailed description of the emulation configuration questions is given in chapter 4.

Notes

8

Using Simulated I/O And Simulated Interrrupts

Configuring Simulated I/O

The simulated I/O subsystem must be set up by answering a series of configuration questions. Your answers to these questions enable simulated I/O, set the control addresses, and define files used for standard I/O.

Detailed information on using simulated I/O with the emulator is provided in the *HP 64000-UX Simulated I/O Reference Manual.*

Modify simulated I/O configuration? yes (no)

no — Answering **no** causes the simulated I/O questions to be skipped. The current simulated I/O configuration is not modified.

yes — Answering **yes** enables you to modify the simulated I/O configuration. The following questions are asked.

Enable polling for simulated I/O? no (yes)

no — Prevents the emulation software from reading the control address for simulated I/O commands. Answering **no** to this question enables you to disable simulated I/O while maintaining the current simulated I/O configuration. Later, when you need to enable simulated I/O, you can do so without having to re-enter control addresses or the file names for standard input, standard output, and standard error out-

put. Answering no also causes the remaining simulated I/O questions to be skipped.

yes — Causes the emulation software to frequently read the control address to determine if the user program has requested any simulated I/O commands. Answering yes causes the following questions to be asked.

Function code data space? none (SUP_DATA) (USR_DATA)

This question asks you to specify the data space where the simio control addresses are located.

If during memory configuration, you specified **modify defined_codes none**, you should use the default answer (**none**) here.

If you specified **modify defined_codes all**, you should select **SUP_DATA** or **USR_DATA** as appropriate for your system.

If you specified **modify defined_codes prog_data**, you should select **USR_DATA**.

Simio control address 1? SIMIO_CA_ONE (<Addr>)
Simio control address 2? SIMIO_CA_TWO (<Addr>)
Simio control address 3? SIMIO_CA_THREE (<Addr>)
Simio control address 4? SIMIO_CA_FOUR (<Addr>)
Simio control address 5? SIMIO_CA_FIVE (<Addr>)
Simio control address 6? SIMIO_CA_SIX (<Addr>)

The symbol SIMIO_CA_ONE is the default symbol associated with the first simulated I/O Control Address. The default symbol may be replaced with any valid symbol or an absolute address. If a symbol is specified, polling of that control address will not begin until a file containing that symbol is loaded. If an absolute address is specified, polling of that address will begin immediately.

The control address must be loaded into memory space assigned as RAM. User programs will run faster if the control address is located in emulation memory. Using target RAM causes the emulator to break into the monitor program every time the control address is polled for simulated I/O commands or data.

The following questions relate to the files associated with the three reserved file names "stdin", "stdout", and "stderr".

File used for standard input? /dev/simio/keyboard (<FILE>)
File used for standard output? /dev/simio/display (<FILE>)
File used for standard error? /dev/simio/display (<FILE>)

The default answers for these questions are "/dev/simio/keyboard", "/dev/simio/display", and "/dev/simio/display" respectively.

These files are not opened until Open (90H) is called with the file names "stdin", "stdout", and "stderr". These files are provided to allow easy redirection of input and output from the keyboard or display to a file or device without modifying the user program. (The compiler standard I/O libraries may open some or all of these reserved files automatically if simulated I/O is used. For more details, see the documentation on the simulated I/O libraries for the compiler you are using.)

Restrictions On Simulated I/O

The two restrictions on the use of simulated I/O are:

- There is a limit of 12 open files at any one time.
- There can only be four active simulated I/O processes at any one time.

Since any simulated I/O file that is opened is associated with a file descriptor, opened files are independent of the control address. Up to 12 files can be opened with a single control address (CA). A total of six control addresses are allowed so that you can execute simulated I/O commands concurrently. Remember, a maximum of 12 simulated I/O files (between the six control addresses) may be open at any one time.

Simulated Interrupts

Simulated interrupts enable you to test software which depends upon the occurrence of preemptive interrupts using out-of-circuit emulation. The simulated interrupt facility is enabled by writing a value of 0ffh to the simulated interrupt control address. The control address is defined during the emulation configuration session. The simulated interrupt facility, when enabled, generates approximately six interrupts per second, depending on what other emulation activities are occurring concurrently, i.e., simulated I/O and display updates.

The simulated interrupt facility can be used to test applications such as a preemptive scheduler in a multitasking system or interrupt driven I/O. Interrupt driven I/O can be simulated by executing simulated I/O commands when a simulated interrupt occurs.

An interrupt is a request by an external device that causes the processor to temporarily suspend normal execution in order to service the interrupting device. Normal execution resumes after the device has been serviced. Interrupts are asynchronous to normal execution. To simulate this action out of circuit, the emulation software running on the host system acts as the external device requesting service.

How Does A Simulated Interrupt Function?

There are only two ways that the emulation software can interrupt the emulator. The first is to reset the processor in the emulator. Since a reset causes the current instruction counter to be lost, continuation of program execution is not possible. Therefore, reset is not usable for simulated interrupts. The second way to interrupt the emulator is to break to the foreground monitor. This is the method used to implement simulated interrupts. Therefore, the emulation monitor must be loaded in order to use simulated interrupts.

The simulated interrupt begins when a value of 0ffh is written into the simulated interrupt control address. The emulation software polls this address just as it polls simulated I/O control addresses. When emulation finds the value 0ffh at the simulated interrupt control address, it causes a break to the emulation monitor. The emulation monitor saves all registers as a normal

part of the monitor entry sequence. The emulation monitor then loops, waiting for a command. The emulation software then sends a simulated interrupt command to the emulation monitor. The simulated interrupt command is a user defined command. The emulation monitor supplied with the emulator contains only a stub which immediately indicates completion.

Note 

You must modify this command to perform whatever action is required when an interrupt occurs. A typical action is a TRAP instruction which vectors to your interrupt handler. See the example program given in figure 8-1. This feature is not available without modifying the monitor. For information on modifying the monitor for simulated interrupts, refer to the section of this chapter entitled "Modifying the Monitor to Use Simulated Interrupts".

Finally, emulation sends the exit monitor command to the emulation monitor. The exit monitor command restores the registers that were saved upon entry to the monitor which causes execution to continue at the point where it was interrupted.

```
*************************************************************************
* This is a simulated interrupt test program.  The vector for           *
* TRAP #14 is pointed to INT_HANDLER.  The SIM_INTERRUPT command        *
* of the monitor must be modified to execute a TRAP #14. Notice         *
* that the SMIINT_CA is enabled, then a delay loop is executed,         *
* then SIMINT_CA is disabled.  INT_HANDLER increments the location      *
* COUNTER to provide a count of the number of interrupts tha occurred.  *
*                        NOTE                                           *
* Simulated interrupts must be enabled in the emulator configuration    *
* and the control address must be set to SIMINT_CA.  To observe the     *
* number of interrupts occuring, use the following command:             *
*                                                                       *
* display memory COUNTER thru COUNTER+7 blocked long repetitively       *
*************************************************************************

                CHIP    68020

                XDEF    START,END1,INT_HANDLER
                XDEF    LOOP,SIMINT_CA,COUNTER

                SECTION INTR_DATA

SIMINT_CA       DC.L    0               ;Set up a memory location to
                                        ; be the control address.
COUNTER         DC.L    0               ;Set up a memory location that
                                        ; the program writes to.

                ORG     038H            ;TRAP #14
                DC.L    INT_HANDLER     ;Notice that the address of the
                                        ; interrupt handler routine is
                                        ; contained in the vector address
                                        ; for a TRAP #14.

                SECTION INTR_PROG

START           MOVE.L  #0,COUNTER      ;Clear the contents of the counter
                                        ; address.
                MOVE.B  #0FFH,SIMINT_CA ;Enable simulated interrupts.
                MOVE.L  #0FFFFFFFH,D0   ;Set up a delay counter value.
LOOP            SUBQ.L  #1,D0           ;Delay for a while.
                BNE     LOOP
                MOVE.B  #0,SIMINT_CA    ;Disable simulated interrupts now.
END1            BRA.B   END1            ;Continuous loop.

INT_HANDLER                             ;This is the interrupt handler
                                        ; routine.
                ADDQ.L  #1,COUNTER      ;Increment the contents of COUNTER.
                RTE                     ;Return from exception.
                END START               ;Define the transfer address so that
                                        ; you may run or step from
                                        ; transfer_address.
```

Figure 8-1. Simulated Interrupt Test Program

Simulated Interrupts Versus Real Interrupts

There are some important differences between simulated interrupts and real interrupts. A simulated interrupt handler must return within a fixed amount of time. Part of the simulated interrupt configuration is the specification of the maximum amount of time that emulation should wait for an interrupt handler to complete execution. If the interrupt handler does not complete within the specified time, emulation forces a break to the monitor and reports a failure to terminate. It is not possible to wait for simulated I/O to complete an interrupt handler.

While the emulation software may appear to be doing several things concurrently, e.g., polling up to six simulated I/O control addresses, polling a simulated interrupt control address, and updating a display, it is in fact only a single HP-UX task performing each of these emulation tasks sequentially. This means that the simulated interrupt must complete before any of the other tasks can begin. That is a motivation for limiting the execution of a simulated interrupt handler to a very short period.

If the handler is permitted to execute for an indefinite period of time, it is possible for the entire emulation program to be 'locked up' by an interrupt handler that is waiting for an event that never occurs.

The final difference between simulated interrupts and real interrupts is that it is not possible for a simulated interrupt to occur while a simulated interrupt is being handled or while the emulator is executing in the monitor.

Simulated Interrupt Configuration

The simulated interrupt facility is not available in real time mode. If real time mode is enabled, the simulated interrupt configuration questions are not asked. When real-time mode is not enabled, the command line displays the following question:

Modify simulated interrupt configuration? no (yes)

Press **Return** for the default (**no**) response.

Press **yes Return** to modify the simulated interrupt configuration.

If you answer **yes**, the simulated interrupt questions will be asked. If you answer **no**, the questions will be skipped. The first question is:

Enable polling for simulated interrupts? no (yes)

no	if no is selected, emulation does not poll a simulated interrupt control address and never causes a simulated interrupt to occur.
yes	if yes is entered, the configuration questions are asked:

Function code data space ? none (SUP_DATA) (USR_DATA

This question asks you to specify the data space where the simulated interrupt control address is located.

If during memory configuration, you specified **modify defined_codes none**, you should use the default answer (**none**) here.

If you specified **modify defined_codes all**, you should select **SUP_DATA** or **USR_DATA** as appropriate for your system.

If you specified **modify defined_codes prog_data**, you should select **USR_DATA**.

Simulated interrupt control address? SIMINT_CA (<Addr)

Enter the value of the simulated control address in response to this question. The value may be a symbolic value or a numeric value. The default is the symbolic value SIMINT_CA.

If you are not linking the emulation monitor program with your target system program, you must be careful when using a symbolic control address such as SIMINT_CA.

The monitor program will store the location of the control address each time that it executes. If you modify your program, and then reload the program without loading the monitor, there is a chance that the symbolic control address will have changed. The monitor program will not recognize a change unless you reload it.

If you do not reload the monitor each time that you load the target system program, you must ORG the control address to a specific location. If you ORG the address, make sure that you modify the

"Simulated interrupt control address" configuration question to point to the new address.

Another solution is to link the monitor program with your program. This causes the monitor to recognize any new address because it loads with your program.

A similar consideration occurs if you modify the control address configuration question. If you are running your program, and then modify the configuration, you must reload your program (and the monitor). Otherwise, the system software does not recognize the new control address and may write to an unknown address.

Maximum delay (in milliseconds) for simulatedinterrupt? 25
(<NUMB>)

The final simulated interrupt configuration question requests the time, in milliseconds, to allow a simulated interrupt handler to execute before assuming that execution of the handler has failed and generates a break to the monitor.

The default time is 25 milliseconds. The default time is approximately equal to the time required to initiate a simulated interrupt and check for its completion on an HP 9000. Even though the resolution of this specification is one millisecond, because of the time that is required to check for completion, the effective resolution is approximately 15 milliseconds. For example, changing the maximum delay from 25 milliseconds to 26 milliseconds probably has no effect on execution. Emulation does not always wait for the maximum delay to pass. If the interrupt handler completes any time before the maximum delay time, emulation forces an immediate return to the interrupted code.

The input to this question is limited to the range of 1 through 10000. Therefore, the maximum delay is 10 seconds. This upper limit was chosen to prevent 'locking up' emulation by an interrupt handler that fails to terminate.

If the user's interrupt handler routine exceeds the maximum delay allowed, the following error message appears on the status line: **"ERROR: Simulated interrupt failed to complete"**.

Modifying The Monitor To Use Simulated Interrupts

The user defined simulated interrupt function allows you to implement interrupt driven code on an emulator which is out of circuit. This command will typically cause a branch to your interrupt handler by means of a TRAP instruction. This command must set the boolean variable SIM_INTS_ENABLED to TRUE and copy the control address to SIM_INT_CA so that the monitor can disable simulated interrupts on entry. If simulated interrupts are not disabled on entry to the monitor, the **break** softkey will not function.

The monitor program must be modified before you can use the simulated interrupt feature. Find the following block of code shown in figure 8-2 in the monitor program.

The TRAP #14 instruction will cause the interrupt routine to be serviced. You must uncomment the instruction or ,if you wish to use a different instruction, you must provide the instruction in the same area of the monitor as the TRAP #14 instruction. If you use another TRAP or different instruction, you must be sure that the routine will be found by the monitor. For example, if you use the TRAP #14 instruction, you must make sure that the address information for your exception routine is in the vector table at address 038h.

When you are finished editing the emulation monitor, be sure to save your changes. It will be necessary to re-assemble and relink the monitor in order to use the simulated interrupts feature.

```
*****************************************************************
*
*  COMMAND 9 ... USER DEFINED SIMULATED INTERRUPT FUNCTION
*
*     THE USER DEFINED SIMULATED INTERRUPT FUNCTION ALLOWS THE USER TO
*     IMPLEMENT INTERRUPT DRIVEN CODE ON AN EMULATOR WHICH IS OUT OF
*     CIRCUIT.  THIS COMMAND WILL TYPICALLY CAUSE A BRANCH TO THE USERS
*     INTERRUPT HANDLER VIA A TRAP INSTRUCTION.  THIS COMMAND MUST SET
*     THE BOOLEAN SIM_INTS_ENABLED TO TRUE AND COPY THE CONTROL ADDRESS
*     TO SIM_INT_CA SO THE MONITOR CAN DISABLE SIMULATED INTERRUPTS ON
*     ENTRY.  IF SIMULATED INTERRUPTS ARE NOT DISABLED ON ENTRY TO THE
*     MONITOR, THE break SOFTKEY WILL NOT WORK.
*
*     THE 64000 WILL SET UP MONITOR_CMD_BUF; SCR_ADDR TO Simulated
*      interrupt control address and issue COMMAND 8009H.
*
*     WHEN THE COMMAND IS COMPLETE, THE 64000 EXPECTS THE  PROCESSOR
*     TO BE IN MONITOR.
*
SIM_INTERRUPT

*   A NON-ZERO VALUE INDICATES THAT SIMULATED INTERRUPTS ARE ENABLED
      MOVE.B    #0FFH,SIM_INTS_ENABLED

*   STORE 0FFH AT SIM_INT_CONTENTS TO KEEP SIMULATED INTERRUPTS ENABLED
      MOVE.B    #0FFH,SIM_INT_CONTENTS

*   STORE THE INTERRUPT CONTROL ADDRESS THAT WAS PASSED BY THE 64000
      MOVE.L    SRC_ADDR,D0
      MOVE.L    D0,SIM_INT_CA

*   INSTRUCTIONS TO BRANCH TO THE USERS INTERRUPT HANDLER GO HERE
*   THIS WILL TYPICALLY BE A TRAP INSTRUCTION.

*     TRAP      #14

      JMP       LOOP_REENTRY

*****************************************************************
```

Notes

9

How The Emulator Works

Overview

This chapter describes how the following emulator functions work:

- The are_you_there monitor function
- The run command
- Software breakpoints
- Single Stepping
- Target memory transfers
- Displaying target memory
- Copying from target memory
- Modifying target memory
- Copying to target memory
- Displaying CPU registers
- Modifying CPU registers

Introduction

The information provided in this chapter will give you a better understanding of how the emulator works and how the emulator interacts with your target system. This information, along with the information provided in chapter 5, Using the Emulator, should help you use the emulator more effectively and avoid problems that can occur when the emulator is used with a target system (in-circuit emulation mode).

Are You There Function?

The "are_you_there" monitor function is the means by which the host computer determines whether or not the 68020 CPU is executing the monitor at a particular time. It is used primarily to display the "running" and "running in monitor" status line messages.

It also performs the important function of checking to see that a break request (level 7 interrupt) resulted in a successful entry to the monitor. The host computer issues break requests for all emulation functions requiring the use of the monitor. If the break fails, the host computer is unable to complete the user specified command, and issues a "cannot break into monitor" message.

The following algorithm describes how the **are_you_there** function works.

1. The host computer writes the value 8000h (bit 15 = 1) to the emulation memory location **MONITOR_CONTROL**.

2. If the emulation monitor is executing, and has completed a previous command, it executes an idle loop. In the idle loop, the monitor is waiting for a user command or for the host to make an "exit monitor" request.

   If the idle loop is executing and **MONITOR_CONTROL** is set to 8000h by the host, the monitor responds by clearing bit 15 (**MONITOR_CONTROL** = 0), and returning to the idle loop.

   If the 68020 CPU is executing in the user program, bit 15 is not cleared, leaving **MONITOR_CONTROL** set to 8000h.

3. The host computer reads emulation memory location **MONITOR_CONTROL**.

If bit 15 of **MONITOR_CONTROL** = 0, the monitor is executing. If bit 15 of **MONITOR_CONTROL** = 1, the user program is executing.

The Run Command

The run command starts execution of your user program. The command allows you to run from a specified address, run until a specified address is executed, or run from a start address until a specified address. The following algorithms describe how the run command is implemented.

Run From Command

When you execute the command "**run from** {**SUPERVISOR_STATE**|**USER_STATE**} <address>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The host verifies that the 68020 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.
3. The host modifies the monitor copy of the return address obtained on entry to the monitor from the level 7 interrupt. It sets the return address to the value specified in the run command.
4. The host modifies the monitor copy of the CPU status register obtained on entry to the monitor from the level 7 interrupt.
   a. If the command specifies "SUPERVISOR_STATE", the host sets the SUPERVISOR/USER bit to 1 (supervisor) so that the 68020 CPU will execute in supervisor mode on exit from the monitor.
   b. If the command specifies "USER_STATE", the host sets the SUPERVISOR/USER bit to 0 (user) so that the CPU 68020 will execute in user mode on exit from the monitor.

5. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to monitor variable **MONITOR_CONTROL**.

6. The host verifies that the 68020 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

Run Until Command

When you execute the command "**run until** <address>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).

2. The host verifies that the 68020 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.

3. The host computer reads the 16-bit word at <address> and saves it internally.

4. The host inserts a BKPT instruction at <address>. The breakpoint is marked internally as a one-shot breakpoint.

5. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to MONITOR_CONTROL.

6. The host verifies that the 68020 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

Run From ... Until Command

When you execute the command "**run from** {**SUPERVISOR_STATE**|**USER_STATE**} <address1> **until** <address2>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).

2. The host verifies that the 68020 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.

3. The host computer reads the 16-bit word at <address2> and saves it internally.

4. The host inserts a BKPT instruction at <address2>. The breakpoint is marked internally as a one-shot breakpoint.

5. The host modifies the monitor copy of the return address obtained on entry to the monitor from the level 7 interrupt. It sets the return address to the value <address1> specified in the run command.

6. The host modifies the monitor copy of the CPU status register obtained on entry to the monitor from the level 7 interrupt.

   a. If the command specifies "SUPERVISOR_STATE", the host sets the SUPERVISOR/USER bit to 1 (supervisor) so that the 68020 CPU will execute in supervisor mode on exit from the monitor.

   b. If the command specifies "USER_STATE",then the host sets the SUPERVISOR/USER bit to 0 (user) so that the CPU 68020 will execute in user mode on exit from the monitor.

7. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to MONITOR_CONTROL.

8. The host verifies that the 68020 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

Software Breakpoints

The following sections describe how the software breakpoint function is implemented in the 68020 emulator. Software breakpoints enable you to enter software breaks into your user program as an aid in debugging your user software. Software breakpoints are also used in the implementation of the run until command.

Note

The exception vector table is referenced only in the case of permanent breakpoints, which make use of the trace exception vector (VBR+24h). If one-shot breakpoints are working correctly, but permanent breakpoints fail, verify that the trace exception vector properly references the monitor (memory location **MONITOR_ENTRY**).

Setting A Software Breakpoint

When you execute the command "**modify sw_breakpoint set {permanent|oneshot}** <bkpt_addr>", the system executes the following algorithm.

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The host computer detects that we actually got to the monitor, issuing an error message "cannot break into monitor" if not.
3. The host gets the 16-bit word at <bkpt_addr> and saves it in ORIG_INST in host system memory.
4. The host inserts the BKPT instruction at <bkpt_addr>.
5. The host initiates a return (RTE) to the user program from the monitor.
6. Host verifies that the emulation monitor was exited, and issues an error message if not.

Executing A Software Breakpoint

When the emulator executes the BKPT instruction specified during emulation configuration, the following events occur:

1. Emulation circuitry detects the occurrence of a BKPT instruction and responds by jamming into the emulation monitor at **SWBK_ENTRY**.

Note 

Only the BKPT instruction specified during emulator configuration is recognized by the emulator.

---

2. The host detects that a breakpoint was executed and issues the message "breakpoint hit at address XXXX."
3. The host restores the original instruction saved in ORIG_INST to <bkpt_addr>.
4. The emulation monitor enters the idle loop, waiting for a user command.

Executing A Run Command After Executing A Software Breakpoint

When you specify a run command after executing a software breakpoint, the following events occur:

"run"

1. The host computer determines if the last BKPT instruction detected is permanent or one-shot.

2. If the breakpoint is one-shot, the emulation monitor returns (RTE) to the user program to begin execution at address BKPT_ADDR.

3. If the breakpoint is permanent, the 68020 CPU is instructed to single-step the instruction at BKPT_ADDR return to the monitor.

4. The host resets the breakpoint and returns (RTE) to the user program as described in steps 2 through 6 of the "Setting A Software Breakpoint" section.

"run from ADDR"

1. The host computer determines if the last BKPT instruction executed was permanent or one shot.

2. If the breakpoint is oneshot, the emulation monitor returns* (RTE) to the user program and begins execution at address ADDR.

3. If the breakpoint is permanent and the "run from" address is set equal to the breakpoint address BKPT_ADDR, the 68020 CPU is instructed to single-step the instruction at BKPT_ADDR and return to the emulation monitor.

4. the host resets the breakpoint as described in steps 2 through 4 of the "Setting A Software Breakpoint" section and then returns* (RTE) to the user program. User program execution begins at ADDR.

*The returns to the user program are accomplished by modifying the stack so that the RTE instruction in the monitor will return to address ADDR, rather than the address originally contained on the stack.

Single Stepping

The following algorithm describes how the single-stepping function is implemented. The single-step function uses the trace exception vector in the exception vector table. If this vector (VBR + 24h) is set incorrectly, single stepping will fail.

When the user executes a step command, the following events occur:

1. The host computer initiates a break to the emulation monitor program by means of a level 7 interrupt.
2. The host computer reads the emulation monitor variable **MONITOR_CONTROL** to verify that the emulator is executing the emulation monitor. If the emulator is not executing in the monitor, the message "cannot break into monitor" is displayed and the step command is aborted.
3. The host instructs the monitor to set the trace bits in the 68020 microprocessor status register (T1 = 1, T0 = 0). This enables the 68020 trace function.
4. If the user specified a "from <address>" the host sets the program counter value on the return stack to <address> so that, upon returning from the monitor to the user program, program execution will begin at <address>.
5. The host initiates a return (RTE) to the user program from the monitor.
6. The 68020 CPU executes a single instruction, and takes the trace exception which reenters the monitor at MONITOR_ENTRY. Note that the trace exception vector (VBR + 24h) must reference MONITOR_ENTRY for this to function correctly.
7. The host verifies that the emulator is executing in the monitor as described in step 2.
8. The host instructs the monitor to clear the trace bits in the 68020 microprocessor status register (T1 = 0, T0 = 0). This disables the 68020 trace function.

9. The emulation monitor enters an idle loop, waiting for a user command.

Target Memory Transfers

The following section describes the process the emulator uses to transfer data to and from target memory. The emulation monitor always attempts to longword align the transfer. Due to the dynamic bus sizing facility of the 68020, this alignment improves total transfer time with 8 and 16-bit memory systems, but is most effective with 32-bit memory systems. This algorithm can be tuned to meet specific target system requirements.

1. At the beginning of the transfer, the monitor examines the lower two bits of the initial target system address to be read from or written to.

   a. If bit 0 of this address is 1, the monitor transfers a single byte to or from the target system using a **MOVES.B** instruction. Following this, the target system address is incremented by one to reflect the next address to be transferred.

   b. If bit 0 of the initial target system address is 0, the byte transfer and address increment does not occur.

   This first step causes the target system address to be aligned to a word address, where bit 0 of the address is 0.

2. The monitor examines bit 1 of the target system address.

   a. If bit 1 of this address is 1, the monitor transfers a single word to or from the target system using a **MOVES.W** instruction. Then, the target system address is incremented by two to reflect the next address to be transferred.

   b. If bit 1 of the initial target system address is 0, the word transfer and address increment does not occur.

   This step aligns the target system address to a longword address, where bits 1 and 0 of the address are 0.

3. The target system address is now longword aligned, i.e., address bits 1 and 0 are both 0. The bulk of the transfer is

then carried out using longword transfers. The operation of the transfer up to this point is summarized in figure 9-1.

Starting Addr Bits 1 and 0	Transfer Description
1 1	a. Copy a byte to longword align b. Increment target address by 1 c. Copy a longword d. Increment target address by 4 e. Repeat steps "c" and "d"
1 0	a. Copy a word to longword align b. Increment target address by 2 c. Copy a longword d. Increment target address by 4 e. Repeat steps "c" and "d"
0 1	a. Copy a byte to word align b. Increment target address by 1 c. Copy a word to longword align d. Increment target address by 2 e. Copy a longword f. Increment target address by 4 g. Repeat steps "e" and "f"
0 0	a. Copy a longword b. Increment target address by 4 c. Repeat steps "a" and "b"

Figure 9-1. Monitor Operation At Start Of Transfer

4. After each longword transfer, the monitor examines the number of bytes remaining in the transfer. If the number is 0, the transfer is complete, and the monitor returns to the idle loop. If the number of bytes remaining to be copied is less than 4 prior to a longword transfer, longword transfers are no longer used, and control passes to monitor code that

finishes up the remaining bytes (3, 2 or 1) of the transaction.

a. If 3 bytes remain, a word transfer followed by a byte transfer is executed.

b. If 2 bytes remain, a single word transfer is performed.

c. If a single byte remains, a byte transfer is used. This monitor function is summarized in figure 9-2.

Number of Bytes Remaining	Transfer Description
4	a. Copy a longword b. Increment target address by 4 c. Return to monitor idle loop
3	a. Copy a word b. Increment target address by 2 c. Copy a byte d. Increment target address by 1 e. Return to monitor idle loop
2	a. Copy a word b. Increment target address by 2 c. Return to monitor idle loop
1	a. Copy a byte b. Increment target address by 1 c. Return to monitor idle loop
0	a. Return to monitor idle loop

Figure 9-2. Monitor Operation At End Of Transfer

Note

The use of the parameters byte, word or long with the display memory command does not alter the target memory transfer algorithm, but provides display formatting control. Similarly, the use of byte, word or long with the modify memory command alters how data is interpreted, but not how the monitor performs the transfer.

Displaying Target Memory

When you execute a display memory command with an address range mapped to target system memory, the emulation monitor reads the specified areas of target memory and copies the memory locations to an internal monitor buffer for transfer to the host computer. This process is described in the following steps:

1. The host computer initiates a break to the monitor (level 7 interrupt).

2. The emulation monitor enters the idle loop, waiting for a host command. The idle loop is located at monitor program symbol MONITOR_LOOP.

3. The host computer detects that the 68020 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message "cannot break into monitor".

4. The host computer writes the memory transfer parameters to designated monitor locations as listed below:

Description	Monitor Location
a. Number of bytes to read	BYTE_COUNT
b. Starting address of target system read	SRC_ADDR
c. Function codes for target system read	SRC_FC
d. Starting address of monitor data buffer write	DST_ADDR
e. Function codes for monitor data buffer write	DST_FC

The monitor data buffer begins at monitor data symbol **MON_XFR_BUF** and is always referenced with the **SUPERVISOR_DATA** function code.

5. The host writes the "read user memory" command (8003H) to **MONITOR_CONTROL**. This causes the monitor to exit the idle loop and begin execution at monitor program symbol **COPY**.

6. The monitor sets up the transfer according to the five parameters listed above, and begins to copy target system memory values to the monitor data buffer using the algorithm described in the previous section. See the emulation monitor listing for additional details. Look at the monitor code following monitor program symbol **COPY**.

7. The host computer detects that the transfer has completed by observing a value of 0000H in **MONITOR_CONTROL**. The host then reads and displays the information in the monitor data buffer. If the display memory command requested a display of more data bytes than the monitor transfer buffer can hold, the host computer sets up a new transfer for the remaining information by repeating the steps beginning with step 4.

8. The host computer initiates a return (RTE) to the user program from the monitor. This occurs as a result of the host writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This operation does not occur if the display memory command was issued while executing in the emulation monitor.

Copying from Target System Memory

The algorithm for copying data from target memory is identical to that used when displaying target memory.

Modifying Target Memory

When you execute a modify memory command with an address mapped to target system memory, the emulation monitor writes to the specified areas of target memory, copying data from the emulation monitor data buffer. The data in the emulation monitor buffer is put there by the host computer. The process for modifying target memory is described in the following steps:

1. The host computer initiates a break to the emulation monitor (a level 7 interrupt).

2. The monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR_LOOP**.

3. The host computer detects that the 68020 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message **"cannot break into monitor"**.

4. The host writes the memory transfer parameters to designated monitor locations as listed below:

Description	Monitor Location
a. Number of bytes to write	BYTE_COUNT
b. Starting address of monitor data buffer read	SRC_ADDR
c. Function codes for monitor data buffer read	SRC_FC
d. Starting address of target system write	DST_ADDR
e. Function codes for target system write	DST_FC

   The monitor data buffer begins at monitor data symbol **MON_XFR_BUF** and is always referenced with the **SUPERVISOR_DATA** function code.

5. The host writes the "write user memory" command (8004H) to **MONITOR_CONTROL**. This causes the monitor to exit the idle loop and begin execution at monitor program symbol **COPY**.

6. The monitor sets up the transfer according to the five parameters listed above, and begins to copy monitor data

buffer values to the target system memory using the target memory transfer algorithm described previously. See the emulation monitor listing for additional details. Look at the monitor code following monitor program symbol **COPY**.

7. the host determines that the transfer has completed by observing a value of 0000H in **MONITOR_CONTROL**. If the modify memory command requested a modify of more data bytes than could be held by the monitor transfer buffer, the host sets up a new transfer for the remaining information by repeating the steps beginning with step 4.

8. The host initiates a return (RTE) to the user program from the monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This operation does not occur if the modify memory command was issued while executing in the emulation monitor.

Copying to Target System Memory

The algorithm for copying data to target system memory is identical to that used when modifying target memory.

Displaying CPU Registers

When you execute a **display registers cpu** command, the following algorithm is executed:

1. The host computer initiates a break to the monitor (a level 7 interrupt).

2. The emulation monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR_LOOP**.

3. The host detects that the 68020 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message "cannot break into monitor". The "are_you_there?" function is used to determine whether or not the monitor is executing.

4. The host reads and displays the register image save area that was constructed on entry into the monitor (i.e. the monitor data area starting with symbol **PCH** and ending with **DFCT**).

5. The host initiates a return (RTE) to the user program from the emulation monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This operation does not occur if the display registers cpu command was issued while executing in the emulation monitor.

Modifying The CPU Registers

When you execute a **modify registers cpu <regname> to <value>** command, the following algorithm is executed:

1. The host computer initiates a break to the emulation monitor (a level 7 interrupt).

2. The monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR_LOOP**.

3. The host detects that the 68020 CPU is executing in the monitor. If the CPU is not executing in the emulation monitor, the host issues the error message "cannot break into monitor". The "are_you_there?" function is used to determine whether or not the emulation monitor is executing.

4. The host writes the modified register value to the corresponding location in the register image save area constructed on entry to the monitor (i.e. the monitor data area starting with symbol **PCH** and ending with **DFCT**).

5. The host initiates a return (RTE) to the user program from the emulation monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This operation does not occur if the modify registers cpu command was issued while the CPU was executing in the monitor.

6. When exiting the monitor, the register image save area is read to reload all CPU registers with their original values on initial entry to the monitor (see monitor program symbol **RTN3**). Since the modify registers command changes values in the register image save area, these new values are loaded in the CPU registers on exit from the monitor.

A

Emulation Error Messages

68020 Emulation Error Messages

This appendix contains a list of 68020 emulation error messages with descriptions of the error and information on how to correct the error, when appropriate. This list describes the most serious emulation errors that you may encounter.

cannot break into monitor

This message is displayed when the host expects to find the CPU executing the monitor, but the "are_you_there?" function indicates otherwise. This message occurs after issuing a command that normally causes a break to the monitor.

If SUPERVISOR_PROG and SUPERVISOR_DATA areas are not overlayed for the emulation monitor, the "are_you_there?" function cannot function properly, resulting in this error message. If function codes are not in use, mapping overlays are not required.

To determine the cause of the failure, setup an analysis trace to trigger on the acknowledge cycle for the level 7 interrupt:

trace trigger_on a= 0ffffffffh s= fcode CPU_SPACE

If the analyzer does not trigger, then it is likely that no level 7 interrupt was generated by the emulator. Check that the "Enable emulator use of INT7?" configuration question has been answered "yes". If so, a hardware error has occurred or the CPU is in a Reset, halt or DMA state (in which case the CPU will not respond to the level 7 interrupt in a timely manner.

The tracelist should show four, 8-bit, emulator generated jam cycles. MONITOR_ENTRY should be the address supplied by these cycles. Compare the tracelist of the monitor entry point to a monitor listing. Determine that the monitor has not been inadvertently overwritten. Be sure that the monitor area is overlayed with SUPERVISOR_DATA and SUPERVISOR_PROGRAM space (not necessary if function codes are turned off).

Check to see that the monitor enters, and stays in the monitor idle loop. If interrupts are enabled in the monitor, an external interrupt routine may be exiting the monitor and not returning properly. Or, if there are frequent interrupts being processed, the "are_you_there?" function may be simply timing out.

Next, setup the analyzer to trigger on the "are_you_there?" monitor command:

trace trigger_on a = MONITOR_CONTROL
d = 8000xxxxH **s = access READ**

The address and data specifications may differ, depending on the address of MONITOR_CONTROL, and the width of the memory system being referenced.

Determine that the "are_you_there?" function in the monitor (ARE_THERE) is functioning properly by observing the trace after capturing the condition where MONITOR_CONTROL is read as 8000H. Compare this trace to the monitor listing.

monitor did not respond to exit request

This message is displayed when the host expects to find the CPU executing somewhere other than in the monitor, but the "are_you_there" monitor function indicates otherwise. This message occurs after issuing a command that results in a return to the user program from the monitor (i.e. display registers while the user program is executing, or "run" while in the monitor, etc.).

If SUPERVISOR_PROG and SUPERVISOR_DATA areas are not overlayed for the emulation monitor, the "are_you_there?" function cannot function properly, resulting in this error message. If function codes are not in use, mapping overlays are not required.

To determine the cause of the failure, setup an analysis trace to trigger on the "exit monitor" command. This can be done with the following trace specification:

trace trigger_on a = MONITOR_CONTROL
d = 8001xxxxH **s = access READ**

Note that the address and data specifications may differ, depending on the address of MONITOR_CONTROL, and the width of the memory system being referenced.

If the monitor is not executing (i.e. in an interrupt routine or elsewhere) at the time of an "exit monitor" command, the command cannot be recognized and this error message will result after a timeout.

Observe the exit mechanism from the monitor, and compare the acquired trace to the monitor listing. Be certain that the monitor has not been overwritten inadvertently.

Once the monitor is exited, check that the user program executes properly. If the user program returns to the monitor immediately after the "exit monitor" command is issued, this message appears.

slow dev at a = XXXX (YY)

This status line message indicates that the CPU is presently attempting to run a bus cycle, but the cycle has not completed after approximately 25 ms. This means that although the CPU asserted address strobe (set it low), the addressed memory (I/O device, etc.) has not yet returned DSACKs, BERR and/or HALT as appropriate.

The XXXX field above indicates the address of the attempted cycle, and the YY field indicates the function code applied to the cycle according to the following table:

SD = Supervisor Data
SP = Supervisor Program
UD = User Data
UP = User Program
R0 = Reserved Address Space 0
R3 = Reserved Address Space 3

R4 = Reserved Address Space 4
CS = CPU Space

Note that this message is simply a warning that the current cycle is taking an unusually long time to complete.

no memory cycles

This status line message indicates that the emulator has not received a low-to-high or high-to-low transition on address strobe for at least 25-30 ms. This message most often appears when executing from cache, if there are no external cycles for long periods of time.

Any device that drives address strobe will inhibit the message, including the emulator 68020, DMA devices, and coprocessors. If a DMA mechanism, for example does not drive address strobe, this message may appear after the specified timeout. (Note that bus cycles where address strobe is not driven cannot be captured by the analyzer.)

This message is simply a warning that address strobes are infrequent.

(no DSACK) message in tracelist

This message normally indicates that a particular CPU cycle was terminated by LBERR or LHALT instead of the usual termination by DSACKs.

This message can also be a clue that the target system is violating the MC68020 specification which specifies that the DSACK signals must not be negated before address strobe is negated by the CPU. This is the case because the analyzer uses a derivative of address strobe as an analysis clock. If DSACKs are high prior to the low-to-high transition of address strobe, a "no DSACK" message can result.

running in monitor

After writing the value 8000H to MONITOR_CONTROL, the host subsequently read MONITOR_CONTROL and received a value of 0000H.

running

After writing the value 8000H to MONITOR_CONTROL, the host subsequently read MONITOR_CONTROL and received a value of 8000H.

Reset (with capital "R")

This message indicates that the CPU is being reset due to the use of the reset softkey in the emulation software.

reset (with lower case "r")

This message indicates that the CPU is being reset by target system hardware.

Attempt to write guarded memory, addr = XXXX

This message appears when an attempt is made to modify a memory location mapped as "guarded" via the "modify memory" command. The offending address is displayed in the XXXX field.

Attempt to read guarded memory, addr = XXXX

This message appears when an attempt is made to display a memory location mapped as "guarded" via the "display memory" command. The offending address is displayed in the XXXX field.

Could not enable breakpoint at address XXXX

This message normally results from attempting to set a breakpoint in target system memory, but for some reason, the emulator could not break into the monitor in order to set the breakpoint. This message also occurs when attempting to set a breakpoint in target ROM, but does not occur when setting a breakpoint in emulation RAM or ROM. Trying to set a breakpoint in a guarded area of memory will also result in this error message.

Could not disable breakpoint at address XXXX

This message normally results from attempting to clear a breakpoint in target system memory, but for some reason, the emulator could not break into the monitor in order to clear the breakpoint.

No breakpoint exists at address XXXX

This message is emitted if the user attempts to clear a breakpoint at an address for which no breakpoint was previously specified. The emulation system is only aware of breakpoints set by the "modify sw_breakpoints set ..." command. If a "modify memory ..." command was used to set the breakpoint, or if the breakpoint existed in the absolute code loaded into the emulator, it is not possible to clear such breakpoints using "modify sw_breakpoints clear ..." commands.

B

Source Files For Getting Started Examples

Introduction

This appendix contains the listings of the "towers.c" and "simint.c" source files. These two source files were compiled and linked with the emulation monitor program to form the towers.X absolute file which was used in all the examples in this manual that show the internal analyzer making trace measurements. The towers.c source listing is first in this appendix. The simint.c source listing is last.

Source File For towers.c

```
/*  LSD:@(#) ..........................................
/*  @(mktid) ..........................................
/*                                                              */
/*  This program demonstrates the solution to the popular       */
/*  "Towers of Hanoi" brain teaser puzzle.  The puzzle consists  */
/*  of 3 pegs and a number of discs of different diameters which */
/*  fit over the pegs.  The discs are ordered by their diameter, */
/*  largest on the bottom,  on one peg.  The object is to move   */
/*  all of the discs from one peg to another such that they end  */
/*  up in the same order on the new peg using the minimum number */
/*  of moves.  Only one disc can be moved at a time, and a larger*/
/*  disc may never be placed on top of a smaller disc.          */
/*                                                              */
/*  The solution can be visualized  using "display simulated_io" */
/*  command.  The number of discs is selected by responding to   */
/*  the input request using the "modify keyboard_to_simio"      */
/*  command and entering a number between 1 and 7.  Multiple    */
/*  numbers separated by spaces can be entered before hitting    */
/*  return to get multiple executions of the program, and "C"    */
/*  may be entered to run the program continuously.             */
/*                                                              */
/*  The speed of the program can be modified in real time with   */
/*  the variable loc_delay and the "modify memory" command.     */
/*                                                              */
/*  NOTE: This file has been designed with the use of "ifdef"    */
/*  to allow it to be compiled and run on the host as well as    */
/*  cross compiled for the emulator.                            */

#include <stdio.h>

#define TRUE 1
#define FALSE0
#define NOVALIDENTRY 1
#define STDOUT 1
#define FIRSTCOL0
#define LEFT 0
#define MIDDLE 1
#define RIGHT 2

#defineMAX_DISC 7
#define MAX_CHARS 16
#define MAX_TOWERS 3
#defineREPEAT 99

/*  If not the AxLS 68020 C compiler, then probably notan ANSI compiler  */
/*  so we must remove the const, volatile, andvoid keywords.             */
#ifndef m68020
#defineconst
#define volatile
#define void
#endif
```

```
typedef struct {
        char disc_char[MAX_CHARS];
      } DISC;
typedef DISC TOWER[MAX_DISC];

static intrun_continuous;
static int num_discs = 4;   /* we will use 4 discs asthe default */
static int move_num;
static intfree_level[MAX_TOWERS];
static int disc_level[MAX_DISC];
staticTOWER display[MAX_TOWERS];

/* This variable may be modified duringemulation to change  */
/* the speed of the program.                                */
const int loc_delay = 500;

static const DISCblank_disc =
        {' ',' ',' ',' ',' ',' ',' ',' ','|','|',' ',' ',' ',' ',' ',' ',' ',' '};

static const TOWER disc_word = {
        {' ',' ',' ',' ',' ',' ',' ',' ','1',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ',' ',' ',' ','2','2',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ',' ',' ','3','3','3',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ','4','4','4','4',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ','5','5','5','5','5','5',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {' ','6','6','6','6','6','6','6',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
        {'7','7','7','7','7','7','7','7',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '}
};
/* Enabling an analysis trace on activity within a function */
/* may not perform properly if the processor prefetches     */
/* the function return (RTS) due to a previous branch.      */
/* Version 2.0 of the HP64903 C Compiler provides a debug   */
/* capability that adds NOP instructions to prevent this    */
/* condition (default options, or -OG should be used).      */
/*                                                          */
/* A less desirable alternative would be to modify the      */
/* source using a dummy statement to provide the padding.   */
/* As an example, a write to the variable "rts_prefetch"    */
/* just before the function return could be used.           */
/* Because the prefetch can be up to 3 words long, we want  */
/* the access address mode to be absolute long and will org */
/* it outside of base page, thus forcing a 3 word opcode.   */

#ifdef m68020

#ifndef DEBUGG
#pragma SECTIONDATA=0x22ff0
        int rts_prefetch;
#pragma SECTIONUNDO
#endif

#else
        intrts_prefetch;
#endif

#ifdef INTERRUPTS
#pragma SECTIONPROG=simint
extern void enable_int();
extern voiddisable_int();
extern int sim_ints_serviced;
```

```
extern intsim_int_ca;
#pragma SECTION UNDO
#endif
/* for local forward referencing */
static void pause();
static void show_discs();
static void init_display();
static void towers();
static int ask_for_number();

main()
{
#ifdef INTERRUPTS
        enable_int();
#endif

         run_continuous = FALSE;

        while ( ask_for_number(&num_discs) == TRUE ) {
#ifdef m68020
                clear_screen(STDOUT);
#endif
                 move_num =0;
                 init_display(LEFT);
                 show_discs();
                 pause();

                towers(num_discs,LEFT,RIGHT,MIDDLE);

                show_discs();

/*  ANSI supports concatenated strings, AxLS simiohas cursor control */
#ifdef m68020
                pos_cursor(STDOUT, FIRSTCOL, num_discs+5);
               printf("\t\tPuzzle with %d discs can be solved in "
                       "%d moves.     \n",num_discs,move_num);
#else
              printf("\n\t\tPuzzle with %d discs can", num_discs);
                printf(" be solved in %d moves.     \n", move_num);
#endif

                 pause();
                 pause();
         }

        return(0);
}

/***** Towers routines *****/

static intask_for_number(num)
int *num;
{
         charerr_char1,err_char2;
         int last_num,ret_val;

        last_num = *num;

         if ( run_continuous == FALSE ) {
```

```
#ifdef m68020
                clear_screen(STDOUT);
                printf("\n\nExecute 'modify keyboard_to_simio' then enter"
                       " one of the following:"
                     "\n\tNumber of discs to use [1-%d]"
                     "\n\t'0'to exit program"
                    "\n\t'C' to run continuouslyusing last number entered\n\n"
                      ,MAX_DISC);
#else
               printf("\n\nEnter one of thefollowing:");
               printf("\n\tNumber of discs to use [1-%d]",MAX_DISC);
               printf("\n\t'0' to exitprogram");
              printf("\n\t'C' to run continuously usinglast number entered\n\n");
#endif

                while (NOVALIDENTRY ) {

                       printf("?");
                        /* scanf will return one of thefollowing three values:
*/
                        /*      1) "0"indicates a scanning error
*/
                       /*      2) "1" indicates valid input
*/
                       /*      3) "EOF"                                          */

                        ret_val = scanf("%d",num);

                        switch (ret_val) {
                               case 0:
                                        err_char1= getchar();
                                        err_char2 =getchar();

                                       /* If a "C" is entered, the last number
will be */
                                       /* used forever (if it was valid). */
                                       if (err_char1 == 'C') {
                                              if ( (last_num < 1) || (last_num
>MAX_DISC) )
                                                      puts(" invalid repeat num-
ber!");
                                              else {
                                                      *num = last_num;
                                                      run_continuous = TRUE;
                                                      return(TRUE);
                                              }
                                       }
/* If the user hits the "suspend" softkey, as a  */
                                        /* courtesy the emulator sends an escape 1
*/
                                        /* character sequence allowing the program
to   */
                                        /* detect that the inputwas suspended.
*/
                                      else if ( (err_char1 == '\033') &&
(err_char2 == '1') )
                                              puts(" input suspended!");

                                        else puts(" input error,re-enter line!");
```

```
                                        fflush(stdin);
                                         /* try again!*/
                                         break;

                                case 1:
                                        if (*num == 0) {
                                              printf(" exiting!\n");
                                               return(FALSE);
                                        }
                                         else if ((*num < 0) ||(*num > MAX_DISC))
                                                printf(" %d is not a valid num-
ber!\n",*num);
                                                /* try again! */
                                         else
                                                return(TRUE);
                                         break;

                                case EOF:
                                        return(FALSE);
                                        break;

                        } /* end switch ret_val*/

                } /* while no valid entry */

        } /* if not run continuous */

        return(TRUE);
}

static void pause()
{
         int i,j;

for (i = 0;i < loc_delay; i++)
                 for (j =0; j < loc_delay; j++) {
                 }

#ifndef DEBUGG
        rts_prefetch = 0;
#endif
}

static void show_discs()
{
        DISC *disc_ptr;
         char *string, *p;
         int disc,tower, ch;

         if ( (p = string = (char *) malloc(BUFSIZ) ) ==NULL ) {
                fputs("malloc failed\n",stderr);
                exit(1);
         }

#ifdef m68020
         pos_cursor(STDOUT, FIRSTCOL,1);
#endif
         for (disc = 0; disc < num_discs; disc++) {
                for (tower = 0; tower < MAX_TOWERS; tower++) {
                        *p++ = '\t';
```

```
                        disc_ptr =&display[tower][disc];
                        for (ch = 0; ch <MAX_CHARS; ch++)
                            *p++ =disc_ptr->disc_char[ch];
                }
               *p++ ='\n';
        }
       fwrite(string,1,(int)(p-string),stdout);

       free(string);

#ifdef m68020
        if (move_num == 0)
           printf("\t----------------\t----------------\t----------------"
                  "\n\t     Peg  0     \t     Peg  1     \t     Peg 2\n"
                    "\n\t\tSolution for Towers with %ddiscs.\n",num_discs);
#else
    printf("\t----------------\t----------------\t----------------\n");
       printf("\t     Peg  0     \t     Peg  1     \t     Peg  2\n");
       if (move_num == 0)
            printf("\n\t\tSolution forTowers with %d discs.\n\n",
                        num_discs);
#endif
}

static void remove_disc(disc,from_peg)
register intdisc,from_peg;
{
        disc--;
       display[from_peg][disc_level[disc]] = blank_disc;
       free_level[from_peg] = disc_level[disc];
}

static voidplace_disc(disc,on_peg)
register int disc,on_peg;
{
       disc--;
        display[on_peg][free_level[on_peg]] =disc_word[disc];
        disc_level[disc] = free_level[on_peg];
       free_level[on_peg]--;

                            }


static void move_disc(i,from,to)
int i,from,to;
{
       move_num++;

        show_discs();

    printf("\n\n\n\n\t\tMove #%d:  Move disk %d from peg %d to",move_num,i,from);
       printf("peg %d     \n",to);

#ifdefINTERRUPTS
        if ( sim_int_ca == -1 )
            printf("\t\t%d simulated interrupts have been serviced.\n"
                        ,sim_ints_serviced);
        else
             printf("\t\tSimulated interrupts have been disabled.      \n");
#endif
```

```
        remove_disc(i,from);

       place_disc(i,to);

        pause();

#ifndef DEBUGG
       rts_prefetch = 0;
#endif
}

static voidinit_display(start_tower)
int start_tower;
{
        inttower,disc;

        /* initialize the display array to be blank*/
        for (tower = 0; tower < MAX_TOWERS; tower++) {
               for (disc = 0; disc < MAX_DISC; disc++)
                       display[tower][disc] = blank_disc;
                free_level[tower] =num_discs - 1;
        }

        /* place num_discs on the
specified tower */
        for (disc = 0; disc < num_discs; disc++){
                display[start_tower][disc] = disc_word[disc];
               disc_level[disc] = disc;
        }
       free_level[start_tower] = 0;
}

static void towers(n,from_peg,to_peg,aux_peg)
register int n,from_peg,to_peg,aux_peg;
{
        if (n == 1)
                move_disc(1,from_peg,to_peg);
        else {
                towers(n-1,from_peg,aux_peg,to_peg);
                move_disc(n,from_peg,to_peg);
                towers(n-1,aux_peg,to_peg,from_peg);
        }
}
```

Source File For simint.c

```
/*  LSD:@(#) 0.04 88/01/19
/*  @(mktid) Unreleased(02.10 05May88)
/*                                                              */
/*  This file contains some very simple examples of routines to  */
/*  use with the simulated interrupt mechanism of the emulator.  */
/*                                                              */

/*  If not the AxLS 68020 C compiler, then probably not an ANSI compiler  */
/*  so we must remove the const, volatile, and void keywords.             */
#ifdef m68020
#pragma SECTION PROG=simint DATA=data CONST=simint
#else
#define const
#define volatile
#define void
#endif

/* This variable records the number of serviced simulated interrupts. */
int sim_ints_serviced = 0;


/* This variable will be used to control simulated interrupts and is */
/* specified in the emulator configuration file.  The host and the   */
/* monitor watch this "control address" to decide whether to perform */
/* the interrupt function or not.  Simulated interrupts will be      */
/* enabled when the control address flag is set to -1 and disabled   */
/* if it is set to 0.  The "modify memory" command can be used to    */
/* enable and disable interrupts in real time once the program has   */
/* has been started. */

/* Remember that using the "volatile" keyword restricts the compiler's */
/* optimization for the entire file, but guarantees proper access of  */
/* variable. */

volatile int sim_int_ca = -1;

void enable_int()
{
        /* enable simulated interrupts from emulator */
        sim_int_ca = -1;
}

void disable_int()
{
        /* disable simulated interrupts from emulator */
        sim_int_ca = 0;
}

#ifdef m68020
#pragma INTERRUPT
static void sim_int_handler()
```

```
{
        /* service simulated interrupts from emulator */
        sim_ints_serviced++;
}

/* Initialize the interrupt vector table to point to our routine. */
#pragma SECTION DATA=0xb8
void (*trap14)() = sim_int_handler;
#pragma SECTION UNDO
#endif
```

C

Timing Comparisons

Introduction

The following tables list timing comparisons between the MC68020RC12, MC68020RC16, MC68020RC20, and MC68020RC25 processors, and the HP 64410C/D emulator.

This Page Intentionally Blank

MC68020RC12/HP 64410 Timing Comparisons

10.5 AC ELECTRICAL SPECIFICATIONS -- CLOCK INPUT

Num	Characteristic	12.5 MHz Min	12.5 MHz Max	HP 64410 Min	HP 64410 Max	Unit
	Frequency of Operation	8	12.5	12.5	25	MHz
1	Cycle Time	80	125	40	80	ns
2,3	Clock Pulse Width	32	87	15	59	ns
4,5	Rise and Fall Times	---	5	---	4	ns

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	12.5 MHz Min	12.5 MHz Max	HP 64410 Min	HP 64410 Max	Unit
6	Clock High to Address Valid	0	40	10*	46*	ns
	Clock High to FC/Size/$\overline{RMC}$ Valid	0	40	11*	47*	ns
6A	Clock High to $\overline{ECS}$, $\overline{OCS}$ Asserted	0	30	9.5*	35*	ns
7	Clock High to Address/Data/FC/$\overline{RMC}$/Size High Impedance	0	80	6.5*	50*	ns
8	Clock High to Address Invalid	0	---	9.5*	---	ns
	Clock High to FC/Size/$\overline{RMC}$ Invalid	0	---	10*	---	ns
9	Clock Low to $\overline{AS}$, $\overline{DS}$ Asserted	3	40	14*	41*	ns
9A[1]	$\overline{AS}$ to $\overline{DS}$ Assertion (Read) (Skew)	–20	20	–8	–12	ns
10	$\overline{ECS}$ Width Asserted	25	---	22	---	ns
10A	$\overline{OCS}$ Width Asserted	25	---	22	---	ns
10B[7]	$\overline{ECS}$, $\overline{OCS}$ Width Negated	20	--	17	---	ns
11[6]	Address Valid to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted, Read)	20	---	15	---	ns
	FC/Size/$\overline{RMC}$ Valid to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted, Read)	20	---	14	---	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$ Negated	0	40	11*	41*	ns
12A	Clock Low to $\overline{ECS}$/$\overline{OCS}$ Negated	0	40	11*	41*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	12.5 MHz Min	12.5 MHz Max	HP 64410 Min	HP 64410 Max	Unit
13	$\overline{AS}$, $\overline{DS}$ Negated to Address Invalid	20	---	0	---	ns
	$\overline{AS}$, $\overline{DS}$ Negated to FC/Size/$\overline{RMC}$ Invalid	20	---	0	---	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted	120	---	137	---	ns
14A	$\overline{DS}$ Width Asserted, Write	50	---	57	---	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	50	---	60	---	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	45	---	51	---	ns
16	Clock High to $\overline{AS}$/$\overline{DS}$/$\overline{RW}$/$\overline{DBEN}$ High Impedance	---	80	---	61*	ns
17[6]	$\overline{AS}$,DS Negated to $\overline{RW}$ High	20	---	17	---	ns
18	Clock High to $\overline{RW}$ High	0	40	10*	41*	ns
20	Clock High to $\overline{RW}$ Low	0	40	10*	41*	ns
21[6]	$\overline{RW}$ High to $\overline{AS}$ Asserted	20	---	18	---	ns
22[6]	$\overline{RW}$ Low to $\overline{DS}$ Asserted (Write)	90	---	98	---	ns
23	Clock High to Data Out Valid	---	40	---	45*	ns
25[6]	$\overline{AS}$,DS Negated to Data Out Invalid	20	---	15	---	ns
25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	20	---	15	---	ns
26[6]	DATA O = Valid to $\overline{DS}$ Asserted (Write)	20	---	13	---	ns
27	Data-In Valid to Clock Low (Data Setup)	10	---	10**	---	ns
27A	Late $\overline{BERR}$ Asserted to Clock Low (Setup Time)	25	---	15**	---	ns
	Late $\overline{HALT}$ Asserted to Clock Low (Setup Time)	25	---	8	---	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$/$\overline{BERR}$ Negated	0	110	0	115	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{HALT}$ Negated	0	110	0	125	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{AVEC}$ Negated	0	110	0	97	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	0	---	6	---	ns
29A	$\overline{DS}$ Negated to Data-In (High Impedance)	---	80	---	46	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	---	60	---	75	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACKx}$ Asserted Skew)	---	20	---	7	ns
32	$\overline{RESET}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	40	0	42*	ns
34	Clock Low to $\overline{BG}$ Negated	0	40	0	44*	ns

C-4 Timing Comparisons

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	12.5 MHz Min	12.5 MHz Max	HP 64410 Min	HP 64410 Max	Unit
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	120	---	57	---	ns
39A	$\overline{BG}$ Width Asserted	120	---	57	---	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	40	10*	41*	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	40	10*	41*	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	40	10*	41*	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	0	40	10*	41*	ns
44[6]	R/$\overline{W}$D Low = to $\overline{DBEN}$ Asserted (Write)	20	---	18	---	ns
45[5]	$\overline{DBEN}$ Width Asserted (Read)	80	---	97	---	ns
	$\overline{DBEN}$ Width Asserted (Write)	160	---	177	---	ns
46	R/$\overline{W}$ Width Asserted (Write or Read)	180	---	217	---	ns
47a	Asynchronous Input Setup Time ($\overline{HALT}$)	10	---	7	---	ns
	Asynchronous Input Setup Time ($\overline{BERR}$, $\overline{DSACKx}$)	10	---	17	---	ns
	Asynchronous Input Setup Time ($\overline{IPLx}$)	10	---	29	---	ns
47b	Asynchronous Input Hold Time ($\overline{HALT}$)	20	---	12	---	ns
	Asynchronous Input Hold Time ($\overline{BERR}$, $\overline{DSACKx}$)	20	---	22	---	ns
	Asynchronous Input Hold Time ($\overline{IPLx}$)	20	---	34	---	ns
48[4]	$\overline{DSACKx}$ = Asserted to $\overline{BERR}$ Asserted	---	35	---	10	ns
	$\overline{DSACKx}$ Asserted to $\overline{HALT}$ Asserted	---	35	---	18	ns
53	Data Out Hold from Clock High	0	---	7*	---	ns
55	$\overline{RW}$ Asserted to Data Bus Impedance Change	40	---	17	---	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	0	---	–3	---	ns
58[10]	$\overline{BGACK}$ = Negated to Bus Driven	1	---	1	---	Clks
59[10]	$\overline{BG}$ = Negated to Bus Driven	1	---	1	---	Clks

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

NOTES:

1. This number can be reduced to 5 nanoseconds if strobes have equal loads.
2. If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted, specification #47 must be met by DSACK0 or DSACK1.
4. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).
5. DBEN may stay asserted on consecutive write cycles.
6. Actual value depends on the clock input waveform.
7. This is a new specification that indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This is a new specification that guarantees operation with the MC68881, which specifies a minimum time for DS negated to AS asserted (specification #13A). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020 does not meet the MC68881 requirements.
9. This is a new specification that allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN.
10. These are new specifications that allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020 regains control of the bus after an arbitration sequence.

HP 64410C/D NOTES:

1. A "*" following a timing value means that the value can be reduced by 2.5 ns if it is a Min spec, or reduced by 6 ns if it is a Max spec if the target clock input into the emulator is not buffered. Timing values followed by "**" can be reduced by 5 ns if the input clock is not buffered. A switch setting inside the emulation pod selects whether the buffered version of the input clock is used by the emulation CPU.

MC68020RC16/HP 64410 Timing Comparisons

10.5 AC ELECTRICAL SPECIFICATIONS -- CLOCK INPUT

Num	Characteristic	16.67 MHz Min	16.67 MHz Max	HP64410 Min	HP64410 Max	Unit
	Frequency of Operation	8	16.67	12.5	25	MHz
1	Cycle Time	60	125	40	80	ns
2,3	Clock Pulse Width	24	95	15	59	ns
4,5	Rise and Fall Times	--	5	---	4	ns

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	16.67 MHz Min	16.67 MHz Max	HP64410 Min	HP64410 Max	Unit
6	Clock High to Address Valid	0	30	10*	46*	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Valid	0	30	11*	47*	ns
6A	Clock High to $\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Asserted	0	20	10*	35*	ns
7	Clock High to Address/Data/FC/$\overline{\text{RMC}}$/Size High Impedance	0	60	7*	50*	ns
8	Clock High to Address Invalid	0	---	10*	---	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Invalid	0	---	11*	---	ns
9	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Asserted	3	30	13*	41*	ns
9A[1]	$\overline{\text{AS}}$ to $\overline{\text{DS}}$ Assertion (Read) (Skew)	-15	15	-8	-12	ns
10	$\overline{\text{ECS}}$ Width Asserted	20	---	12	---	ns
10A	$\overline{\text{OCS}}$ Width Asserted	20	---	12	---	ns
10B[7]	$\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Width Negated	15	---	7	---	ns
11[6]	Address Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	15	---	5	---	ns
	FC/Size/$\overline{\text{RMC}}$ Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	15	---	4	---	ns
12	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Negated	0	30	10*	41*	ns
12A	Clock Low to $\overline{\text{ECS}}$/$\overline{\text{OCS}}$ Negated	0	30	10*	41*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	16.67 MHz		HP64410		Unit
		Min	Max	Min	Max	
13	$\overline{AS}$, $\overline{DS}$ Negated to Address Invalid	15	---	0	---	ns
	$\overline{AS}$, $\overline{DS}$ Negated to FC/Size/$\overline{RMC}$ Invalid	15	---	0	---	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted	100	---	97	---	ns
14A	$\overline{DS}$ Width Asserted, Write	40	---	37	---	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	40	---	40	---	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	35	---	31	---	ns
16	Clock High to $\overline{AS}$/$\overline{DS}$/$\overline{RW}$/$\overline{DBEN}$ High Impedance	---	60	---	61*	ns
17[6]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{RW}$ High	15	---	7	---	ns
18	Clock High to $\overline{RW}$ High	0	30	10*	41*	ns
20	Clock High to $\overline{RW}$ Low	0	30	10*	41*	ns
21[6]	$\overline{RW}$ High to $\overline{AS}$ Asserted	15	---	8	---	ns
22[6]	$\overline{RW}$ Low to $\overline{DS}$ Asserted (Write)	75	---	68	---	ns
23	Clock High to Data Out Valid	---	30	---	45*	ns
25[6]	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	15	---	5	---	ns
25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	15	---	5	---	ns
26[6]	Data Out Valid to $\overline{DS}$ Asserted (Write)	15	---	3	---	ns
27	Data-In Valid to Clock Low (Data Setup)	5	---	10*	---	ns
27A	Late $\overline{BERR}$ Asserted to Clock Low (Setup Time)	20	---	15**	---	ns
	Late $\overline{HALT}$ Asserted to Clock Low (Setup Time)	20	---	8	---	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$/$\overline{BERR}$ Negated	0	80	0	75	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{HALT}$ Negated	0	80	0	85	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{AVEC}$ Negated	0	80	0	57	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	0	---	6	---	ns
29A	$\overline{DS}$ Negated to Data-In (High Impedance)	---	60	---	46	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	---	50	---	55	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACKx}$ Asserted Skew)	---	15	---	7	ns
32	$\overline{RESET}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	30	0	42*	ns
34	Clock Low to $\overline{BG}$ Negated	0	30	0	44*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	16.67 MHz Min	16.67 MHz Max	HP64410 Min	HP64410 Max	Unit
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	90	---	57	---	ns
39A	$\overline{BG}$ Width Asserted	90	---	57	---	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	30	10*	41*	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	30	10*	41*	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	30	10*	41*	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	0	30	10*	41*	ns
44[6]	$\overline{RW}$ Low to $\overline{DBEN}$ Asserted (Write)	15	---	8	---	ns
45[5]	$\overline{DBEN}$ Width Asserted (Read)	60	---	67	---	ns
	$\overline{DBEN}$ Width Asserted (Write)	120	---	127	---	ns
46	$\overline{RW}$ Width Asserted (Write or Read)	150	---	157	---	ns
47a	Asynchronous Input Setup Time ($\overline{HALT}$)	5	---	7	---	ns
	Asynchronous Input Setup Time ($\overline{BERR}$, $\overline{DSACKx}$)	5	---	17	---	ns
	Asynchronous Input Setup Time ($\overline{IPLx}$)	5	---	29	---	ns
47b	Asynchronous Input Hold Time ($\overline{HALT}$)	15	---	12	---	ns
	Asynchronous Input Hold Time ($\overline{BERR}$, $\overline{DSACKx}$)	15	---	22	---	ns
	Asynchronous Input Hold Time ($\overline{IPLx}$)	15	---	34	---	ns
48[4]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$ Asserted	---	30	---	10	ns
	$\overline{DSACKx}$ Asserted to $\overline{HALT}$ Asserted	---	30	---	18	ns
53	Data Out Hold from Clock High	0	---	7*	---	ns
55	$\overline{RW}$ Asserted to Data Bus Impedance Change	30	---	17	---	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	0	---	–3	---	ns
58[10]	$\overline{BGACK}$ Negated to Bus Driven	1	---	1	---	Clks
59[10]	$\overline{BG}$ Negated to Bus Driven	1	---	1	---	Clks

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

NOTES:

1. This number can be reduced to 5 nanoseconds if strobes have equal loads.
2. If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted, specification #47 must be met by DSACK0 or DSACK1.
4. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).
5. DBEN may stay asserted on consecutive write cycles.
6. Actual value depends on the clock input waveform.
7. This is a new specification that indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This is a new specification that guarantees operation with the MC68881, which specifies a minimum time for DS negated to AS asserted (specification #13A). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020 does not meet the MC68881 requirements.
9. This is a new specification that allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN.
10. These are new specifications that allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020 regains control of the bus after an arbitration sequence.

HP 64410C/D NOTES:

1. A "*" following a timing value means that the value can be reduced by 2.5 ns if it is a Min spec, or reduced by 6 ns if it is a Max spec if the target clock input into the emulator is not buffered. Timing values followed by "**" can be reduced by 5ns if the input clock is not buffered. A switch setting inside the emulation pod selects the buffered or unbuffered version of the target clock to be used by the emulation CPU.

MC68020RC20/HP 64410 Timing Comparisons

10.5 AC ELECTRICAL SPECIFICATIONS -- CLOCK INPUT

Num	Characteristic	20 MHz		HP64410		Unit
		Min	Max	Min	Max	
	Frequency of Operation	12.5	20	12.5	25	MHz
1	Cycle Time	50	80	40	80	ns
2,3	Clock Pulse Width	20	54	15	59	ns
4,5	Rise and Fall Times	---	5	---	4	ns

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	20 MHz		HP64410		Unit
		Min	Max	Min	Max	
6	Clock High to Address Valid	0	25	10*	46*	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Valid	0	25	11*	47*	ns
6A	Clock High to $\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Asserted	0	15	9*	35*	ns
7	Clock High to Address/Data/FC/$\overline{\text{RMC}}$/Size High Impedance	0	50	6*	50*	ns
8	Clock High to Address Invalid	0	---	10*	---	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Invalid	0	---	11*	---	ns
9	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Asserted	3	25	13*	41*	ns
9A[1]	$\overline{\text{AS}}$ to $\overline{\text{DS}}$ Assertion (Read) (Skew)	–10	10	–8	–12	ns
10	$\overline{\text{ECS}}$ Width Asserted	15	---	7	---	ns
10A	$\overline{\text{OCS}}$ Width Asserted	15	---	7	---	ns
10B[7]	$\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Width Negated	10	---	2	---	ns
11[6]	Address Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	10	---	0	---	ns
	FC/Size/$\overline{\text{RMC}}$ Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	10	---	0	---	ns
12	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Negated	0	25	11*	41*	ns
12A	Clock Low to $\overline{\text{ECS}}$/$\overline{\text{OCS}}$ Negated	0	25	11*	41*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	20 MHz		HP64410		Unit
		Min	Max	Min	Max	
13	$\overline{AS}$, $\overline{DS}$ Negated to Address Invalid	10	---	0	---	ns
	$\overline{AS}$, $\overline{DS}$ Negated to FC/Size/$\overline{RMC}$ Invalid	0	---	0	---	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted	85	---	77	---	ns
14A	$\overline{DS}$ Width Asserted, Write	38	---	27	---	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	38	---	30	---	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	30	---	23	---	ns
16	Clock High to $\overline{AS}$/$\overline{DS}$/$\overline{RW}$/$\overline{DBEN}$ High Impedance	---	50	---	61*	ns
17[6]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{RW}$ High	10	---	2	---	ns
18	Clock High to $\overline{RW}$ High	0	25	10*	41*	ns
20	Clock High to $\overline{RW}$ Low	0	25	10*	41*	ns
21[6]	$\overline{RW}$ High to $\overline{AS}$ Asserted	10	---	3	---	ns
22[6]	$\overline{RW}$ Low to $\overline{DS}$ Asserted (Write)	60	---	53	---	ns
23	Clock High to Data Out Valid	---	25	---	45*	ns
25[6]	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	10	---	0	---	ns
25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	10	---	0	---	ns
26[6]	Data Out Valid to $\overline{DS}$ Asserted (Write)	10	---	0	---	ns
27	Data-In Valid to Clock Low (Data Setup)	5	---	10**	---	ns
27A	Late $\overline{BERR}$ Asserted to Clock Low (Setup Time)	15	---	15**	---	ns
	Late $\overline{HALT}$ Asserted to Clock Low (Setup Time)	15	---	8	---	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$/$\overline{BERR}$ Negated	0	65	0	55	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{HALT}$ Negated	0	65	0	65	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{AVEC}$ Negated	0	65	0	47	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	0	---	6	---	ns
29A	$\overline{DS}$ Negated to Data-In (High Impedance)	---	50	---	46	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	---	43	---	45	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACKx}$ Asserted Skew)	---	10	---	7	ns
32	$\overline{RESET}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	25	0	42*	ns
34	Clock Low to $\overline{BG}$ Negated	0	25	0	44*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	20 MHz		HP64410		Unit
		Min	Max	Min	Max	
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	75	---	57	---	ns
39A	$\overline{BG}$ Width Asserted	75	---	57	---	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	25	10*	41*	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	25	10*	41*	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	25	10*	41*	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	0	25	10*	41*	ns
44[6]	$\overline{RW}$ Low to $\overline{DBEN}$ Asserted (Write)	10	---	3	---	ns
45[5]	$\overline{DBEN}$ Width Asserted (Read)	50	---	42	---	ns
	$\overline{DBEN}$ Width Asserted (Write)	100	---	92	---	ns
46	$\overline{RW}$ Width Asserted (Write or Read)	125	---	127	---	ns
47a	Asynchronous Input Setup Time ($\overline{HALT}$)	5	---	7*	---	ns
	Asynchronous Input Setup Time ($\overline{BERR}$, $\overline{DSACKx}$)	5	---	17	---	ns
	Asynchronous Input Setup Time ($\overline{IPLx}$)	5	---	29	---	ns
47b	Asynchronous Input Hold Time ($\overline{HALT}$)	15	---	12	---	ns
	Asynchronous Input Hold Time ($\overline{BERR}$, $\overline{DSACKx}$)	15	---	22	---	ns
	Asynchronous Input Hold Time ($\overline{IPLx}$)	15	---	34	---	ns
48[4]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$ Asserted	---	20	---	10	ns
	$\overline{DSACKx}$ Asserted to $\overline{HALT}$ Asserted	---	20	---	18	ns
53	Data Out Hold from Clock High	0	---	7	---	ns
55	$\overline{RW}$ Asserted to Data Bus Impedance Change	25	---	17	---	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	0	---	-3	---	ns
58[10]	$\overline{BGACK}$ Negated to Bus Driven	1	---	1	---	Clks
59[10]	$\overline{BG}$ Negated to Bus Driven	1	---	1	---	Clks

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

NOTES:

1. This number can be reduced to 5 nanoseconds if strobes have equal loads.
2. If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted, specification #47 must be met by DSACK0 or DSACK1.
4. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).
5. DBEN may stay asserted on consecutive write cycles.
6. Actual value depends on the clock input waveform.
7. This is a new specification that indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This is a new specification that guarantees operation with the MC68881, which specifies a minimum time for DS negated to AS asserted (specification #13A). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020 does not meet the MC68881 requirements.
9. This is a new specification that allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN.
10. These are new specifications that allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020 regains control of the bus after an arbitration sequence.

HP 64410C/D NOTES:

1. A "*" following a timing value means that the value can be reduced by 2.5ns if it is a Min spec, or reduced by 6ns if it is a Max spec if the target clock input to the emulator is not buffered. Timing values followed by "**" can be reduced by 5ns if the input clock is not buffered. A switch setting inside the emulation pod selects whether the buffered or unbuffered version of the target clock is used by the emulation CPU.

MC68020RC25/HP 64410 Timing Comparisons

10.5 AC ELECTRICAL SPECIFICATIONS -- CLOCK INPUT

Num	Characteristic	25 MHz		HP64410***		Unit
		Min	Max	Min	Max	
	Frequency of Operation	12.5	25	12.5	25	MHz
1	Cycle Time	40	80	40	80	ns
2,3	Clock Pulse Width	15	59	15	59	ns
4,5	Rise and Fall Times	---	4	---	4	ns

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	25 MHz		HP64410***		Unit
		Min	Max	Min	Max	
6	Clock High to Address Valid	0	25	10*	46*	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Valid	0	25	11*	47*	ns
6A	Clock High to $\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Asserted	0	15	10*	35*	ns
7	Clock High to Address/Data/FC/$\overline{\text{RMC}}$/Size High Impedance	0	40	7*	50*	ns
8	Clock High to Address Invalid	0	---	10*	---	ns
	Clock High to FC/Size/$\overline{\text{RMC}}$ Invalid	0	---	11*	---	ns
9	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Asserted	3	20	14*	41*	ns
9A[1]	$\overline{\text{AS}}$ to $\overline{\text{DS}}$ Assertion (Read) (Skew)	-10	10	-8	-12	ns
10	$\overline{\text{ECS}}$ Width Asserted	10	---	7	---	ns
10A	$\overline{\text{OCS}}$ Width Asserted	10	---	7	---	ns
10B[7]	$\overline{\text{ECS}}$, $\overline{\text{OCS}}$ Width Negated	5	---	2	---	ns
11[6]	Address Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	5	---	0	---	ns
	FC/Size/$\overline{\text{RMC}}$ Valid to $\overline{\text{AS}}$ Asserted (and $\overline{\text{DS}}$ Asserted, Read)	5	---	0	---	ns
12	Clock Low to $\overline{\text{AS}}$, $\overline{\text{DS}}$ Negated	0	20	11*	41*	ns
12A	Clock Low to $\overline{\text{ECS}}$/$\overline{\text{OCS}}$ Negated	0	20	11*	41*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	25 MHz		HP64410***		Unit
		Min	Max	Min	Max	
13	$\overline{AS}$, $\overline{DS}$ Negated to Address Invalid	5	---	0	---	ns
	$\overline{AS}$, $\overline{DS}$ Negated to FC/Size/$\overline{RMC}$ Invalid	5	---	0	---	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted	65	---	102	---	ns
14A	$\overline{DS}$ Width Asserted, Write	30	---	67	---	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	30	---	27	---	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	25	---	23	---	ns
16	Clock High to $\overline{AS}$/$\overline{DS}$/$\overline{RW}$/$\overline{DBEN}$ High Impedance	---	40	---	61*	ns
17[6]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{RW}$ High	5	---	2	---	ns
18	Clock High to $\overline{RW}$ High	0	20	10*	41*	ns
20	Clock High to $\overline{RW}$ Low	0	20	10*	41*	ns
21[6]	$\overline{RW}$ High to $\overline{AS}$ Asserted	5	---	2	---	ns
22[6]	$\overline{RW}$ Low to $\overline{DS}$ Asserted (Write)	45	---	42	---	ns
23	Clock High to Data Out Valid	---	25	---	45*	ns
25[6]	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	5	---	0	---	ns
25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	5	---	0	---	ns
26[6]	Data Out Valid to $\overline{DS}$ Asserted (Write)	5	---	0	---	ns
27	Data-In Valid to Clock Low (Data Setup)	5	---	10**	---	ns
27A	Late $\overline{BERR}$ Asserted to Clock Low (Setup Time)	10	---	15**	---	ns
	Late $\overline{HALT}$ Asserted to Clock Low (Setup Time)	10	---	8	---	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$/$\overline{BERR}$ Negated	0	50	0	75	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{HALT}$ Negated	0	50	0	85	ns
	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{AVEC}$ Negated	0	50	0	57	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	0	---	6	---	ns
29A	$\overline{DS}$ Negated to Data-In (High Impedance)	---	40	---	46	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	---	32	---	27	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACKx}$ Asserted Skew)	---	10	---	7	ns
32	$\overline{RESET}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	20	0	42*	ns
34	Clock Low to $\overline{BG}$ Negated	0	20	0	44*	ns

10.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc ±5%; GND = 0 Vdc; TA = 0 to 70 C)

Num	Characteristic	25 MHz Min	25 MHz Max	HP64410*** Min	HP64410*** Max	Unit
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	60	---	57	---	ns
39A	$\overline{BG}$ Width Asserted	60	---	57	---	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	20	10*	41*	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	20	10*	41*	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	20	10*	41*	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	0	20	10*	41*	ns
44[6]	$\overline{RW}$ Low to $\overline{DBEN}$ Asserted (Write)	5	---	3	---	ns
45[5]	$\overline{DBEN}$ Width Asserted (Read)	40	---	77	---	ns
	$\overline{DBEN}$ Width Asserted (Write)	80	---	117	---	ns
46	$\overline{RW}$ Width Asserted (Write or Read)	100	---	137	---	ns
47a	Asynchronous Input Setup Time ($\overline{HALT}$)	5	---	7	---	ns
	Asynchronous Input Setup Time ($\overline{BERR}$, $\overline{DSACKx}$)	5	---	17	---	ns
	Asynchronous Input Setup Time ($\overline{IPLx}$)	5	---	29	---	ns
47b	Asynchronous Input Hold Time ($\overline{HALT}$)	10	---	12	---	ns
	Asynchronous Input Hold Time ($\overline{BERR}$, $\overline{DSACKx}$)	10	---	22	---	ns
	Asynchronous Input Hold Time ($\overline{IPLx}$)	10	---	34	---	ns
48[4]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$ Asserted	---	15	---	10	ns
	$\overline{DSACKx}$ Asserted to $\overline{HALT}$ Asserted	---	15	---	18	ns
53	Data Out Hold from Clock High	0	---	7*	---	ns
55	$\overline{RW}$ Asserted to Data Bus Impedance Change	20	---	17	---	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	0	---	-3	---	ns
58[10]	$\overline{BGACK}$ Negated to Bus Driven	1	---	1	---	Clks
59[10]	$\overline{BG}$ Negated to Bus Driven	1	---	1	---	Clks

MC68020 electrical specifications reprinted courtesy Motorola, Inc.

NOTES:

1. This number can be reduced to 5 nanoseconds if strobes have equal loads.
2. If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to asserted, specification #47 must be met by DSACK0 or DSACK1.
4. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).
5. DBEN may stay asserted on consecutive write cycles.
6. Actual value depends on the clock input waveform.
7. This is a new specification that indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This is a new specification that guarantees operation with the MC68881, which specifies a minimum time for DS negated to AS asserted (specification #13A). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020 does not meet the MC68881 requirements.
9. This is a new specification that allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN.
10. These are new specifications that allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020 regains control of the bus after an arbitration sequence.

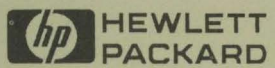
HP 64410C/D NOTES:

1. A "*" following a timing value means that the value can be reduced by 2.5ns if it is a Min spec, or reduced by 6ns if it is a Max spec if the target clock input into the emulator is not buffered. Timing values followed by "**" can be reduced by 5ns if the input clock is not buffered. A switch setting inside the emulation pod selects whether the buffered or unbuffered version of the target clock is used by the emulation CPU.

   When the emulator is operated with a target clock of 25Mhz, the emulation hardware guarantees that one wait-state gets added into the emulation CPU's bus cycle. The timing values in the above tables reflect this additional wait-state.

Index

Notes



64410-90903
E0688
Printed In U.S.A. 06/88