DataGeneral

Customer Documentation

Intelligent Local Area Network Controller Programmer's Reference

Intelligent Local Area Network Controller Programmer's Reference

014-000796-01

Ordering No. 014-000796

Printed in the United States of America
Rev. 01, December 1987

Notice

Intelligent Local Area Network
Controller
Programmer's Reference
014-000796-01

Revision History:

Original Release - September, 1984
First Revision - December, 1987

The content in this revision is unchanged from 014-000796-00. This revision changes only the printing and binding details.

Preface

This is the programmer's reference manual for the Data General Corporation Intelligent Local Area Network Controller (ILC). The manual describes the logical and physical components of the ILC, and the commands and instructions that operate it.

This manual is intended for experienced assembly language programmers and field engineers. We assume that this audience is familiar with Data General Corporation assembly language, peripheral programming techniques, and hardware.

Manual Organization

Chapter 1, General Information, contains general information about the ILC, including a functional and physical description of the controller.

Chapter 2, Host/Controller Programmed Input/Output Interface, describes the programming interface between the host and the ILC microECLIPSE® processor. This chapter includes a description of the virtual I/O register set, signals, serial protocol, message transmission, and message interpretation.

Chapter 3, Data Channel, discusses the ILC data channel facility and the local mapping facility.

Chapter 4, System Control Port Programming, explains the control and status commands associated with the the System Control Port; and describes the Real Time Clock and Watch Dog Timer.

Chapter 5, Data Mover, is a description of the function and programming of the Data Mover device.

Chapter 6, microECLIPSE Processor/Intel 82586 Interface, provides information about the function and programming of the Intel 82586 Local Area Network Communications Controller (LANCC).

Chapter 7, I/O Interface, describes the function and programming of the Fujitsu MB502A Input/Output device.

Chapter 8, System Initialization, contains a description of ILC operation from Hyperspace, and the sequence of events leading to and during system initialization.

Appendix A, Instruction Set, is a dictionary of the assembly language commands that are supported by the ILC.

Appendix B, Device and Interrupt Code Assignments, lists device and interrupt codes.

Appendix C, Signal Lists, contains lists of internal and external cable signals, and backpanel connections.

Appendix D, Sample Programs, contains examples of ILC programs.

Related Manuals

The assembly language instruction set for ILC microECLIPSE system programming is described in detal in the *ECLIPSE® S/20 and C/30 Assembly Language Programmer's Reference* (014-000682).

The contents of the accumulators used in the data channel map upload are described in *ECLIPSE MV/4000® System Functional Characteristics* (014-000736).

The following manuals should be consulted for details about the Ethernet, and the Intel and Fujitsu components that are part of the ILC.

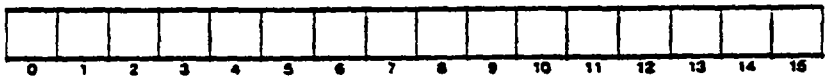
- *The Ethernet*, Version 2.0, Digital Equipment Corp., Intel Corp., and Xerox Corp., November, 1982.
- *IEEE Specification P802.3/83/0.01/D*, and addendum, May 4, 1983. Institute of Electrical and Electronic Engineers.
- *82586 Reference Manual*, Intel Corp., January 1983 (order number: 210891-001).
- Fujitsu MB502A Encoder/Decoder data sheet.

Typesetting Conventions

The conventions we use for instruction formats are as follows:

Where	Means
COMMAND	Uppercase boldface type indicates mnemonics for commands and associated symbols. They must be entered as shown, except that you can use either upper- or lowercase letters.
argument	Lowercase italics indicate required arguments for which numbers or symbols must be substituted.
{*argument_1* | *argument_2*}	Lowercase italics enclosed in braces indicate that you have a choice of arguments for which numbers or symbols must be substituted. You must enter one of the arguments. Do not enter the braces; they merely set off the choice.
[option]	Lowercase italics enclosed in square brackets indicate optional arguments. Do not enter the brackets; they merely set off the choice.

The following diagram shows the arrangement of the bits in a word or address. Unless otherwise specified, the bits are numbered in ascending order from left to right, with the Most Significant Bit (MSB) on the left.



MSB **LSB**

Note that in Chapter 6 the bits are numbered in descending order from left to right. This is done to conform with the numbering used in the Intel *82586 Reference Manual*. In either case, the bits are read from left to right, and the MSB is always on the left.

Contacting Data General Corporation (DGC)

- If you have any comments about this manual, please return the Reader's Comment form provided at the end of the manual.
- DGC manuals can be ordered from Data General Corporation by contacting:

  Technical Information and Publications Service (TIPS)
  Educational Services - MS F019
  Data General Corporation
  4400 Computer Drive Westboro, MA 01580
  (617) 366-8911 ext. 4032

- If you have software problems, please notify your local DGC systems engineer.
- If you have hardware problems, please notify the DGC Field Engineering Dispatch Center.

Contents

Chapter 1 - General Information

Chapter 2 - Host/Controller Programmed Input/Output Interface

Chapter 3 - Data Channel

Chapter 4 - System Control Port Programming

Chapter 5 - Data Mover Facility

Chapter 6 - microECLIPSE Processor/Intel 82586 Interface

Chapter 7 - I/O Interface

Chapter 8 - System Initialization

Appendix A - Instruction Set

Appendix B - Device and Interrupt Code Assignments

Appendix C - Signal Lists

Appendix D - Sample Programs

Chapter 1
General Information

The Intelligent Local Area Network Controller (ILC) is an Input/Output (I/O) device that manages the movement of data between an Ethernet™/IEEE 802 and a host computer. The ILC uses half-duplex, synchronous transmission, and provides Carrier Sensing Multiple Access/Collision Detecting (CSMA/CD) channel access. The ILC is user-programmable, and supports serial transfer of data. The ILC/host interface includes both a data channel facility, and Programmed Input/Output (PIO) instructions. The host machine can be any ECLIPSE® MV/class machine.

Functional Description

Figure 1.1 is a block diagram of the ILC. Functionally, the ILC can be split into three parts:

- A microECLIPSE® system that consists of the microECLIPSE CPU, complete access to both ILC 64-page memory banks, A and B, a full ECLIPSE data channel facility, and access to programs in Hyperspace memory.
- A Link device that includes a Data Mover and an Intel 82586 Local Area Network Communications Controller (LANCC). Control information and buffer space for this device are located in ILC memory bank B.
- An I/O device that includes a Fujitsu MB502A encoder/decoder, and provides the connection to the Ethernet/IEEE 802 transceiver.

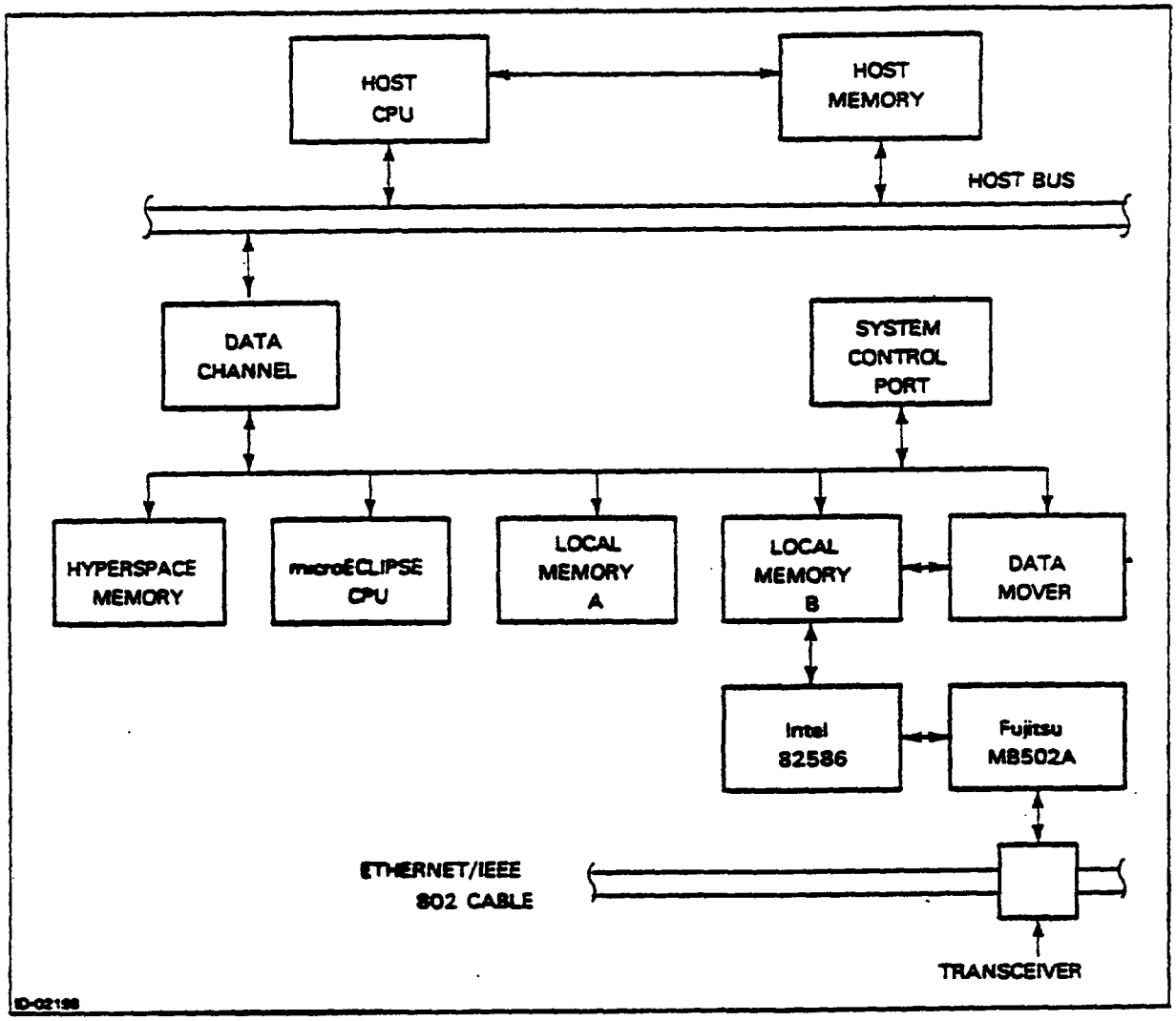


Figure 1-1. ILC Block Diagram

The microECLIPSE System

The host/controller interface is provided by the microECLIPSE CPU. The microECLIPSE system can access host memory directly through the data channel, and can exchange status and control information with the host CPU using PIO commands. The PIO interface is discussed in Chapter 2; the data channel is discussed in Chapter 3.

The microECLIPSE system contains support logic that provides the microECLIPSE CPU with the following functions and devices:

- An internal mapping facility
- A Real Time Clock facility
- A System Control Port
- A Watch Dog Timer.

The microECLIPSE system can read or write to either of the 64-page memory banks, A or B. The ILC internal mapping facility manages microECLIPSE system access to host memory, and to both local memory banks. It also lets the ILC upload the host map, and provides write-protection for each location accessed in memory. These aspects of the mapping facility are described in detail in Chapter 3.

The microECLIPSE system includes the use of Hyperspace RAM and PROM. Hyperspace features are discussed in Chapter 8.

The Link Device

Data transfer between a host and an Ethernet/IEEE 802 takes place in two steps. A block of data is moved from the host to ILC memory bank B. The block is then moved from memory bank B to the Ethernet. The same two steps can occur in the opposite direction.

The transfer between memory bank B and the host is accomplished by the Data Mover. The transfer of data between memory bank B and the Ethernet is accomplished by the Intel 82586 LANCC.

The Data Mover is programmed by the microECLIPSE CPU. Some commands are issued to the Data Mover directly, while others are issued via the System Control Port. Once programmed, the Data Mover can manage the movement of data between the host and memory bank B, freeing the microECLIPSE system to perform higher level functions. The Data Mover is described in detail in Chapter 5.

The 82586 LANCC uses a System Control Block and PIO commands to manage the transfer of blocks of data. The Link device is initialized and directed by the microECLIPSE CPU. This is discussed in detail in Chapter 6.

Input/Output

The Input/Output interface with the transceiver consists of a Fujitsu MB502A encoder/decoder. The MB502A provides Manchester encoding of data and the carrier sensing facility. The MB502A also transmits packet collision information that has been collected by the tranceiver to the 82586 LANCC. The Input/Output device is discussed in Chapter 7.

Physical Description

The ILC is a standard fifteen-inch, multi-layer printed circuit board that resides in the host, occupying either an I/O-Memory or an I/O-only slot. A maximum of two ILCs can be hosted on a machine. The ILC is connected to the host convenience panel with an internal cable, and from the convenience panel to the transceiver with an external cable. One pair of internal/external cables is required per controller. The combined length of the cables is a maximum of 50 meters. DGC supplies either a 5-meter or a 20-meter external cable.

Signal lists for the interface between the I/O-Memory or I/O-only slot and the controller's bus; and specifications and signal lists for the internal and external cables can be found in Appendix C.

The ILC uses volages of +5V and +12V, with respective power consumptions of +5V at a maximum of 6 amps, and +12V at a maximum of 600 milliamps.

ILC Device Code

You select an Intelligent LAN Controller (ILC) I/O device code by setting DIP switch 1 on the board. This may be any device code from 0 to 77, and is set on a bit-by-bit basis. The DIP switch is made up of six individual switches that select six different bits: DS0 (most significant bit) through DS5 (least significant bit). The bits are set from left to right as follows:

SW 6	SW 5	SW 4	SW 3	SW 2	SW 1
DS0	DS1	DS2	DS3	DS4	DS5

where SW denotes the individual switch.

An open switch is equivalent to a 1 in the device code. A closed switch is equivalent to a 0.

You should set the device code before the board is installed.

Test Features and Diagnostics

The ILC runs a program at powerup that tests the kernel microECLIPSE system logic and ROM. The controller also supports a host-driven diagnostic that includes a microECLIPSE CPU system exercise of support devices and of the communications channel.

The ILC can be tested in two loopback modes. The first mode is a function of Intel 82586 LANCC programming, and tests the data path between ILC RAM and the LANCC. The second mode is a function of System Control Port programming, and tests the data path from the ILC RAM through the Fujitsu MB502A and back.

Chapter 2
Host/Controller Programmed Input/Output Interface

This chapter describes the Programmed Input/Output (PIO) interface between the Intelligent LAN Controller (ILC) and the host. This interface is a subset of the ECLIPSE PIO interface, and includes the unidirectional signals START (S), CLEAR (C), and PULSE (P), as well as six virtual PIO registers. The signals and registers are described below.

Virtual Input/Output (VIO) Register Set

The ILC uses a set of I/O registers as part of its host interface. These registers and the PIO status flags let the host transfer command and status information between it and a device. The ILC has three status flags: the standard Busy and Done flags, and an additional Special flag.

The controller's PIO registers are *virtual registers* that exist in host memory rather than on the controller. To effect PIO commands you must direct the host to select an area in host memory to be used as memory space for the ILC virtual registers, and to put the address of that host memory space in the virtual registers. The host then directs the controller to retrieve the register contents from host memory, and to begin execution.

Standard Data General Corporation assembler instructions such as LDA, STA, or BLM pass information in and out of these virtual registers. The VIO registers, shown in Figure 2.1, correspond to the standard PIO registers DOA, DOB, DOC, DIA, DIB, and DIC.

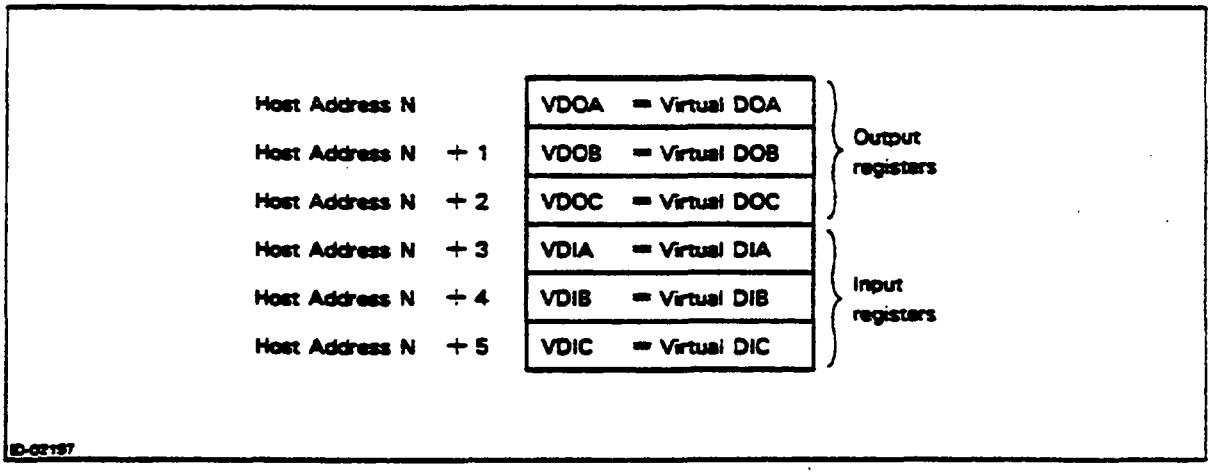


Figure 2-1. Virtual Registers

The starting address for these registers is programmable. Use the commands described later in this chapter to specify and download the initial address to the controller. Once the controller receives the initial address, it responds to VIO commands under the control of a program in its Hyperspace PROM.

The terms "input" and "output" are always used in relation to the host. Input registers contain information that the controller has sent to the host. Output registers contain information that the host has sent to the controller.

Host/microECLIPSE System Interface Signals

Table 2.1 lists the host/microECLIPSE system interface signals. These signals provide a cross-interrupt facility between the ILC microECLIPSE system and the host, and a serial communications channel. The protocol of the serial communications channel is described in the next section.

Table 2-1. Signal Definitions

Signal Name	Effect on VIO flags		
	Busy	Done	Special
CLEAR or IORST (C)	clear	clear	clear
START (S)	set	—	clear
PULSE (P)	—	clear	set

If you want to send a signal, use an NIO instruction. The format of this instruction is as follows:

NIO[*f*] *ac,device*

Where:

f is the type of signal you want to send:

CLEAR = C
START = S
PULSE = P

ac is the host accumulator

device is the device code for the controller.

For example, to clear the Busy and Done flags of a controller with a device code of 36, enter:

NIOC 0,36

Serial Protocol

All device flag manipulation and monitoring is done by System Control Port programming (see Chapter 4, "System Control Port Programming").

Data exchange in either direction is controlled by a synchronized handshake between the host and the device. No explicit framing bits are required.

Message Transmission

The host transmits data to the controller by sending signals that affect the controller's Busy and Done flags. The following steps are involved in transmission of a data bit from the host to the controller:

1. When the controller is ready to receive a data bit it clears its Busy flag and sets its Done flag.
2. The controller then monitors its Done flag, waiting for the flag to be cleared by the host.
3. The host monitors the controller's Done flag. When the host sees that the controller's Done flag is set, the host transmits a data bit.
4. To send a data bit of 1, the host sends an S signal, setting the controller's Busy flag. This action is followed by a P signal from the host. The P signal clears the controller's Done flag, signaling the controller that a data transmission has occurred.
5. To send a data bit of 0, the host sends no signal that affects the Busy flag (the Busy flag remains clear). The host sends only a P signal. This action clears the controller's Done flag, signaling the controller that a data transmission has occurred.

You can send data bits to the controller using the **NIOS** and **NIOP** instructions: these send the S and P signals. This protocol is used to transmit a 21-bit command word to the controller. The command word directs the controller to prepare for VIO commands.

Send the command word starting with bit 0, and ending with bit 20. When it has received all 21 bits, the controller interprets and executes the command word. The controller then signals that it is ready for the next command by setting its Done flag.

Mask Bit Assignment

The ILC uses mask bit 12. When this bit is set, the controller cannot interrupt the host.

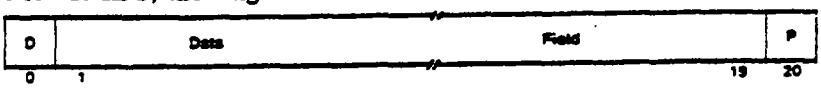
When interrupts are not masked, the controller is able to signal the host whenever there is new data in the virtual input registers.

When interrupts are masked, the host must monitor the controller's Done flag continuously. It can do this by using a programmed "**SKP** on non-zero bit" command (**SKPDN** or **SKPDZ**) to the ILC to find out when new data is available.

Interrupts can be cleared with a **NIOP** instruction to the controller.

Message Interpretation

For the ILC, the length of the command word is 21 bits. The format is as follows:



Bit 0	Diagnostic	Aids in board fault diagnosis. SET (1): The 19-bit data field is interpreted as a VIO address command. CLEAR (0): the ILC performs a diagnostic user RAM test that identifies any hard bit failures in either of the 64-page banks of local RAM (bank A or bank B). If a failure in the local ILC memory is detected, the microECLIPSE system halts. If no failure is detected, the controller clears its Busy flag, sets its Done flag, and re-enters its serial receive state.
Bits 1-19	Data Field	Define the starting logical address of the virtual register block in host memory. Bits 1-4 define the data channel map to be used. Bits 5-9 specify the page in the data channel map. Bits 10-19 specify the word address on the page. The controller sets the start of the VIO block to the specified 19-bit address, leaves the initial serial state and waits for the receipt of a START (S) signal, indicating that the virtual data-out registers (VDOA, VDOB, and VDOC) contain valid command information.
Bit 20	Parity	Maintains even parity over the entire message. This bit must be calculated by the user and included in the serial message. If a parity error occurs, the controller resets its Done flag and hangs. The host times out and resets the controller with a CLEAR (C) signal.

Serial Link Interpretation After a Reset

The controller can be reset by an IORST signal or a CLEAR (C) signal sent from the host. Either signal causes the controller to clear all three PIO flags (Busy, Done, and Special) and causes the microECLIPSE system to enter a self-test/initialization program in Hyperspace PROM. This program is described in Chapter 8, "System Initialization."

When the microECLIPSE system finishes its initialization routine, it enters an internal polling state. In this state it continuously monitors its Busy and Special flags. If the host sends a PULSE (P) signal (Special flag is set), the controller enters the serial-receive state and signifies its readiness to receive data by setting its Done flag. A START (S) signal (Busy flag is set) from the host is interpreted by the microECLIPSE system as a request for a program load from the ILC to the host. Programming for this function is located in Hyperspace PROM, and is explained in Chapter 8, "System Initialization."

Instructions

Memory reference instructions are used to transfer control and status information between the host and the controller via the virtual register set.

The **NIOS** instruction sends a Start signal to the controller. Registers VDOA and VDOB contain command-specific information. Register VDOC always contains a command operation code.

The **LDA** instruction retrieves status information from the controller. The information in registers VDIA, VDIB, and VDIC depends on the command sent to the controller and whether an error occurred during command execution.

Instruction Syntax

Use the following syntax to code an **LDA**, **NIO**, or **STA** instruction.

instruction_name[*f*] *ac*,{*device* }
{*location*}

Where:

instruction_name	Is the instruction you want to issue.
f	Is the signal you want to issue: C = CLEAR S = START P = PULSE
ac	Is the host accumulator that contains the transfer information: 0 = AC0 1 = AC1 2 = AC2 3 = AC3
device	Is the device code of the controller. We used ILC as the assembler mnemonic for the controller's device code. Set the device code with DIP switch 1, as described in the *Device Code* section in Chapter 1.
location	Specifies a memory location for an **LDA** or **STA** instruction only.

Hyperspace Command Interpretation

At system initialization, the microECLIPSE system is forced to execute programs located in Hyperspace memory (see Chapter 8, "System Initialization"). The controller's Hyperspace PROM includes self-test and boot support programs. These programs support a number of unique interpretations of the VIO register block. This is done to help the system user download operational code, and to identify the controller.

Once operating code has been loaded, you can define your own virtual register or control block scheme, not confined to the Hyperspace format.

The commands that you enter into the VDOC register help the host program the ILC; they are used to identify the controller, to set up the VIO registers, and to tell the ILC the initial address of the VIO registers. While the microECLIPSE system is executing from Hyperspace it responds to the command codes in VDOC as described below. The commands are shown in Table 2.2.

Table 2-2. Command Codes

Command Code	Command Description
0	GET INFORMATION
3	RUN
5	LOAD HOST ADDRESS
6	DUMP BLOCK
11	LOAD BLOCK

If you specify a code not listed in Table 2.2, the microECLIPSE system posts an error condition, copies the contents of the input registers into the output registers, and sets bit 0 in the VDIC register to 1. No command execution takes place.

Note these restrictions when writing to the virtual registers:

- The host cannot change the contents of the VIO output registers while the host Busy flag is set.
- The ILC cannot change the contents of the VIO input registers while its own Done flag is set.

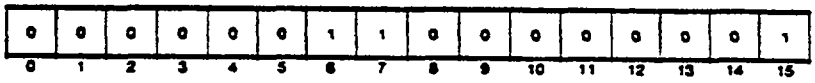
GET INFORMATION *(command code 0)*

This command returns the ILC identification code via the VDIA register. The following table shows the register contents for this command.

Table 2-3. Register Contents for GET INFORMATION Command

Output Registers	Content
VDOA	Not used
VDOB	Not used
VDOC	0
Input Registers	**Content**
VDIA	Controller identification code and hardware revision model code
VDIB	Not used
VDIC	0

On completion of this command, the contents of VDIA are:



Bits 0 - 7	Identification Code	Identify this board as the 15-inch Ethernet/IEEE 802 synchronous controller. The unique code is 00000011_2, or 003_8.
Bits 8 - 15	Revision or Model	Define the revision or model code. These bits should not be changed unless hardware revisions affect software operation. The current revision code for this controller is 00000001_2, or 001_8.

There are no error returns for this command.

This command instructs the ILC to start executing at the ILC RAM location specified in the VDOA register.

If the microECLIPSE system maps have been programmed in such a way that the address in the VDOA register is on a data channel page, the microECLIPSE system executes code located in host memory, using the host data channel facility. The following table shows the register contents for this command.

Table 2-4. Register Contents for RUN Command

Output Registers	Content
VDOA	Address in local RAM (ILC memory bank A or B) where execution is to begin NOTE: Bit 0 of the VDOA register has no effect for this command.
VDOB	Not used
VDOC	3_8
Input Registers	**Content**
VDIA	Same as VDOA
VDIB	Not used
VDIC	3_8

There are no error returns for this command.

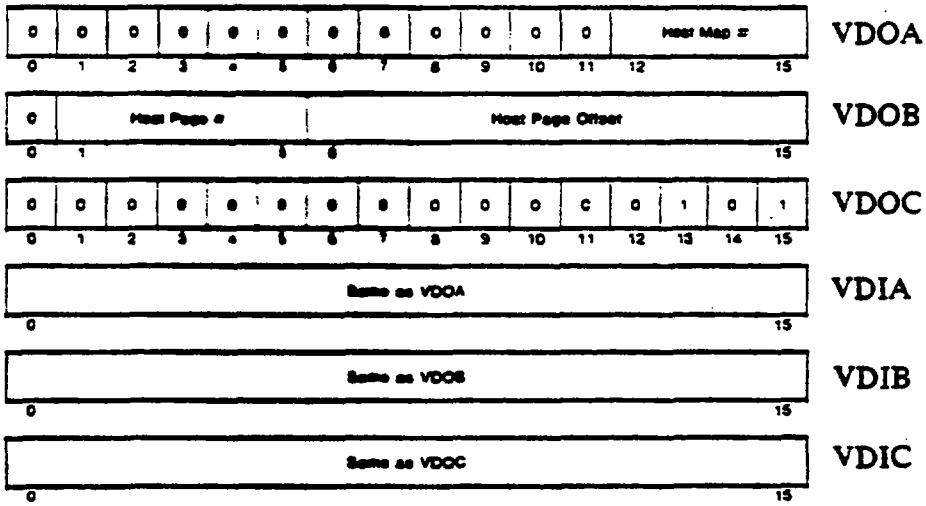
LOAD HOST ADDRESS *(command code 5)*

This command specifies the starting address of a destination block in host memory for a **DUMP BLOCK** command, or the starting address of a source block in host memory for a **LOAD BLOCK** command. The following table shows the register contents for this command.

Table 2-5. Register Contents for LOAD HOST ADDRESS Command

Output Registers	Content
VDOA	Host map number
VDOB	Host page and page offset information
VDOC	5_8
Input Registers	**Content**
VDIA	Same as VDOA
VDIB	Same as VDOB
VDIC	Same as VDOC

On completion of this command, the contents of the virtual input/output registers should be:



VDOA **Bits 0-11** Not used, and must be zero.

Bits 12-15 Contain the host map number. This number defines which of sixteen possible host data channel maps will be used in the subsequent **DUMP BLOCK** or **LOAD BLOCK** command execution.

VDOB **Bit 0** Not used, and must be zero.

Bits 1-5 Contain the host map page number.

Bits 6-15 Contain the page offset number.

The 4-bit host map number, together with the 5-bit map page number from the ILC mapping facility, and the 10-bit offset number, form a 19-bit host logical address. See Chapter 3, "Data Channel," for more information about the map page number.

VDOC	Bits 0-12	Not used, and must be zero.
	Bits 12-15	Contains the command code $101_2 = 5_8$.
VDIA	Bits 0-15	Are the same as VDOA bits 0 - 15.
VDIB	Bits 0-15	Are the same as VDOB bits 0 - 15.
VDIC	Bits 0-15	Are the same as VDOC bits 0 - 15.

DUMP BLOCK *(command code 6)*

This command moves an ILC memory block into host memory. The following table shows the register contents for this command.

Table 2-6. Register Contents for DUMP BLOCK Command

Output Registers	Content
VDOA	Starting logical address of the ILC memory block to be moved
VDOB	Word count of the block to be moved
VDOC	6 (octal)
Input Registers	**Content**
VDIA	Same as VDOA
VDIB	Same as VDOB
VDIC	Same as VDOC

If bit 0 of VDOA is SET(1), hyperspace memory is dumped; if VDOA bit 0 is CLEAR(0), the address is interpreted as an ILC memory bank A or B address.

You must specify the starting address of the destination block in host memory with a **LOAD HOST ADDRESS** command before issuing this command.

LOAD BLOCK *(command code 11)*

This command moves a host memory block into microECLIPSE system local memory. The following table shows the register contents for this command.

Table 2-7. Register Contents for LOAD BLOCK Command

Output Registers	Content
VDOA	Starting logical address of the ILC user memory block to which the source (host) block is to be moved
VDOB	Word count of the block in the ILC local memory to which the source (host) block is to be moved
VDOC	11_8
Input Registers	**Content**
VDIA	Same as VDOA
VDIB	Same as VDOB
VDIC	Same as VDOC

If bit 0 of VDOA is SET(1), hyperspace memory is loaded; if VDOA bit 0 is CLEAR(0), the address is interpreted as an ILC memory bank A or B address.

You must specify the starting address of the source block in host memory with a **LOAD HOST ADDRESS** command before issuing this command.

Chapter 3
Data Channel

The Intelligent LAN Controller (ILC) supports the ECLIPSE data channel bus, and provides full ECLIPSE data channel facilities. In addition to the standard channel cycles, the ILC has two channel cycles that are supported on ECLIPSE MV/Family computers only: map upload, and read-modify-write.

The first section of this chapter describes the ILC local mapping facility. This facility lets the microECLIPSE processor access 128 pages of local memory. In addition, the mapping facility directs access to host memory over the data channel.

The second and third sections describe the ILC special data channel cycles: upload of the host mapping facility, and read-modify-write.

Local Mapping Facility

A program can select the portions of physical memory that correspond to logical addresses by writing to the ILC local map.

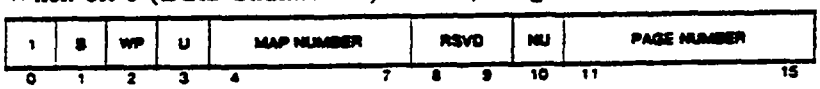
The map does the following:

1. Lets the microECLIPSE processor access all 128 pages of ILC local memory
2. Maintains two separate ILC logical memory spaces (system and user)
3. Provides fast context switching between system and user space (system call)
4. Performs automatic context switching upon receipt of an interrupt
5. Selects local logical pages that are physically located in host memory
6. Initiates a data channel map upload operation (if the ILC is installed on an ECLIPSE MV/Family computer)
7. Provides write-protection for ILC logical pages that are physically located in both local and host memory.

Operation of the Local Mapping Unit

The most significant bit of the map word, bit 0, determines whether bits 1-15 are interpreted as an address that is accessed over the data channel, or an address in ILC local memory.

When bit 0 (Data Channel bit) is SET, assignments of bits 1-15 are as follows:



Bit 1	Bank B	Not used when bit 0 is set.
Bit 2	Write Protect	SET(1): write-protect a page of logical memory. A write-protect violation results in a jump to Hyperspace, and execution of Hyperspace PROM code.
Bit 3	Upload	SET(1): The map number and physical page number are used to identify a map slot on a host map that is being reprogrammed by the ILC in an upload. This is explained in more detail later in this chapter. CLEAR(0): The map number and physical page number are used to identify a map slot on a host map. The contents of this map slot, combined with the 10-bit page offset from the logical address, identifies a physical location in host memory.
Bits 4-7	Map Number	Identify one of the 16 possible host data channel maps assigned to a particular logical page of ILC memory. These bits are active high bits.
Bits 8-9	Reserved	Not used, and must be 0.
Bit 10	Not Used	When bit 0 is set, indicating a data channel transaction, this bit is not used. It is a "don't care" bit.
Bits 11-15	Page Number	These five active-high bits specify the map slot, or word, on the selected map. When host memory is being accessed (bit 3 is clear), these bits, combined with the map number, identify a map slot on a host map. The address information provided by this map slot, together with the ten-bit logical address, form the physical address.

When bit 0 (Data Channel bit) is CLEAR, the bit assignments of bits 1-15 are as follows:

Bit 1	Bank B	SET(1): Logical page address is physically located in ILC memory bank B. CLEAR(0): Logical page address is physically located in ILC memory bank A.
Bit 2	Write Protect	SET(1): Write-protect a page of logical memory. A write-protect violation results in a jump to Hyperspace, and execution of Hyperspace PROM code.
Bit 3	Upload	Not used when bit 0 is clear.
Bits 4-7	Map Number	Not used when bit 0 is clear.
Bits 8-9	Reserved	Not used, and must be 0.
Bits 10-15	Page Number	Provide translation from logical to physical address. These six bits, together with the 10-bit logical address, form the physical address.

Physical Implementation

The ILC mapping unit, or map, consists of 1024 by 16 bit RAM. The first 32 words are the system map; the next 32 words are the user map. The ILC is always in either the system or the user map.

The map must always be written via store word commands such as **STA** or **BLAM**. Unlike the rest of the user-accessible memory on the controller, the map is not byte-writable.

Map Access

Access to the map is possible only via logical page zero, and is performed on a memory-mapped input/output basis. Logical memory locations $400\text{-}477_8$ represent map RAM locations $0\text{-}63_8$.

Both the system page zero (RAM location 0) and user page zero (RAM location 32) can be mapped to any physical page or assigned to a host page that is accessed via the data channel.

Context Switch

ILC local memory is logically divided into system and user memory. The following events cause a context switch from user to system:

An interrupt	The interrupt handler resides in system context. An interrupt to the ILC switches the microECLIPSE processor to system context, disables interrupts, and starts a standard ECLIPSE interrupt sequence.
A System Call (SYS instruction)	A system call switches the microECLIPSE processor to system context, pushes a return block on the stack, and transfers control to the system call handler via location 2.

The following events cause a context switch from system to user:

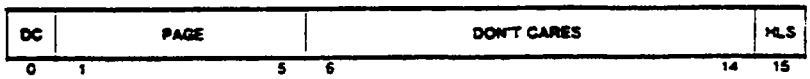
A **MAPON** instruction	A **MAPON** instruction causes an immediate switch from system to user context. Execution of a **MAPON** instruction in user context is treated as a **NOP**.
Setting the MAP bit in the map status register	A set MAP bit causes a context switch after the next **POPB, POPJ, RTN,** or **RSTR** instruction that does not cause a stack fault; or after the first level of the next indirect memory reference.

Map Upload

To perform a map upload, you must program the local map word shown earlier in the *Operation of the Local Mapping Unit* section in this chapter. The map word must be programmed in the following way:

- Bit 0 (Data Channel bit) must be SET(1), indicating that this is a data channel transaction.
- Bit 1 is a "don't care" bit.
- Bit 2 can be SET or CLEAR, and determines whether the transaction is write-protected.
- Bit 3 (Upload bit) must be SET(1), indicating that this is an Upload operation.
- Bits 4-7 specify the host map to be modified.
- Bits 8-9 are not used, and must be zero
- Bit 10 is a "don't care" bit.
- Bits 11-15 specify the slot (or word) on the host map that is to be modified.

After the local map has been programmed, the upload can take place using the following format for the target address:



Bit 0	Don't Care	Not used during a map upload. If a memory transaction other than an upload is specified by the local map, this bit functions as an indirect bit. See Appendix A for an explanation of the indirect bit. You can do an indirect upload of the host map, but you must use a local map word that has the Upload bit set to zero.
Bits 1-5	Page	This field is the page number of the ILC local logical address. These bits are used to address the local mapping unit. They identify the local map word that you programmed.
Bits 6-14	Don't Care	In the map upload process, these bits are not used. The value assigned to them has no significance.
Bit 15	HLS	SET(1): Indicates that the low order portion of the slot is to be loaded by this operation. CLEAR(0): Indicates that the high order portion of the slot is to be loaded by this operation.

Once the local mapping unit has been programmed and the target address determined for the upload operation, two data transfers must be made over the host data channel in order to write the high and low portions of the host map unit.

The content of the accumulators used as the sources of the data for these two operations is as described in the DCH MAP reference section of the *ECLIPSE MV/4000® System Functional Characteristics* (014-000736)

Read-Modify-Write Facility

The ILC provides hardware support for indivisible data channel read-modify-writes (RMW) of host memory locations. The supported microECLIPSE instructions are **ISZ** and **DSZ**.

The RMW facility functions only when code is fetched or executed locally within the ILC; it does not function when code is being fetched or executed over the data channel.

RMW uses the data channel on MV/Family machines. An attempt to use the RMW facility on any other type of machine results in an undefined microECLIPSE system state.

Chapter 4
System Control Port Programming

The System Control Port (SCP) lets the microECLIPSE CPU control various registers, flags and devices of the Intellegent LAN Controller (ILC).

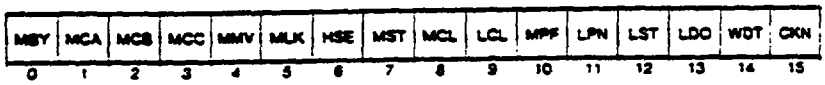
The SCP provides the microprocessor with:

1. Cross communication with the host CPU
2. Start/stop and interrupt/interrupt-masking control of the Real Time Clock facility, the Data Mover, and the Link system
3. Start/stop and interrupt/interrupt-masking status monitoring of the Real Time Clock facility, the Data Mover, and the Link system
4. Control and status of the following microECLIPSE system facilities:
   - Enter Hyperspace
   - Write Protection
   - Power fail
   - Host system resets
   - Controller identification
   - Visual indicator
   - Watch Dog Timer (WDT).

The next section of this chapter shows you the bit assignments for SCP programming. The second section describes the Real Time Clock, and the third describes the Watch Dog Timer facility.

Commands Providing Control and Status

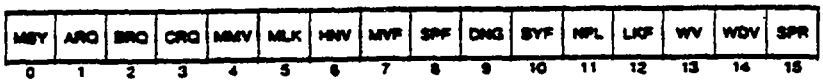
DOA *ac,50*



Bit 0	Mask Busy	SET(1): Masks an interrupt request resulting from the Busy flag being set. The Busy flag can be set by the microECLIPSE processor (using System Control Port programming) or by the host (by sending a START signal to the ILC).
Bit 1	Mask RTCA	SET(1): Masks an interrupt request from the 5.2 ms Real Time Clock.
Bit 2	Mask RTCB	SET(1): Masks an interrupt request from the 52 ms Real Time Clock.
Bit 3	Mask RTCC	SET(1): Masks an interrupt request from the 520 ms Real Time Clock.
Bit 4	Mask Mover Device	SET(1): Masks an interrupt request caused by the Done flag setting for the Data Mover device (interrupt device 57).
Bit 5	Mask Link Device	SET(1): Masks an interrupt request caused by the Done flag setting for the Link device (interrupt device 51).
Bit 6	Hyperspace Enter	SET(1): Causes a Non-Maskable Interrupt (NMI) to the microECLIPSE processor and causes the ILC to enter Hyperspace code execution area.
Bit 7	Mover Start	SET(1): Starts Data Mover transfers of blocks of data. Block move parameters should be set up before you set this bit. See Chapter 5, "Data Mover," for more information.
Bit 8	Mover Clear	SET(1): Clears a done request from the Data Mover device. See Chapter 5 for more information.
Bit 9	Link Clear	SET(1): Clears a done request from the Link device. See Chapter 6 for more information.
Bit 10	Mask Power Fail	SET(1): Masks an interrupt request resulting from a host system power failure. This bit is set after system power up, or when the host sends a IORST or CLEAR signal to the controller.
Bit 11	ESI Loopback Enable	SET(1): Enables a Link device loopback. This bit provides a means to select the internal loopback facility for the Fujitsu MB502A encoder/decoder. The loopback function lets you test the transmit- and receive-data paths.

Bit 12	Link Start	SET(1): Causes a Link device channel attention (**LNK START**). See Chapter 6 for more information.
Bit 13	Diagnostic LED	SET(1): Turns on an indicator lamp that is visible from the front plane of the controller, indicating pass/no-pass status. It is enabled after the system initialization self-test, and after diagnostic completion.
Bit 14	Watch Dog Timer	SET(1): Enables the Watch Dog Timer facility. See the last section of this chapter for more information.
Bit 15	RTC Control	SET(1): Lets the Real Time Clocks interrupt the microECLIPSE processor unless RTC interrupts have been disabled by setting bits 1, 2 or 3.

DIA *ac,50*



Bit 0	Mask Busy	SET(1): Indicates that the BUSY interrupt is masked.
Bit 1	RTCA Request	SET(1): Indicates that an RTC A (5.2 ms) interrupt is valid.
Bit 2	RTCB Request	SET(1): Indicates that an RTC B (52 ms) interrupt is valid.
Bit 3	RTCC Request	SET(1): Indicates that an RTC C (520 ms) interrupt is valid.
Bit 4	Mask Mover	SET(1): Indicates that interrupts from the Data Mover device are masked.
Bit 5	Mask Link	SET(1): Indicates that interrupts from the Link device are masked.
Bit 6	Hyper Enter - No Violation	SET(1): Indicates entry into Hyperspace code execution because of assertion of the HSE bit (**DOA** *ac*,50; Bit 6).
Bit 7	Mover Done Flag	SET(1): Indicates that the Data Mover Done flag is set. CLEAR(0): Indicates that the Data Mover Done flag is clear.
Bit 8	Special Flag	Shows the state of the ILC Special flag (SET or CLEAR), and is set by System Control Port programming, or by the PIO PULSE (P) signal from the host.
Bit 9	Done Flag	Shows the state of the host Done flag (SET or CLEAR). This bit is set by System Control Port programming.
Bit 10	Busy Flag	Shows the state of the ILC Busy flag (SET or CLEAR). The bit is set by System Control Port programming, or by the PIO START (S) signal from the host.
Bit 11	Network Program Load	SET(1): Indicates that the ILC has been hardware selected to respond to requests initiated by the network for a program load of the host processor.
Bit 12	Link Done Flag	Shows the state of the Link device Done flag (SET or CLEAR).
Bit 13	Write Violation	Shows the state of the Write Violation Status flag. After the microECLIPSE processor writes to a write-protected page, this bit becomes latched. It can be cleared only by an AIORST instruction from the microECLIPSE processor or by a RESET or CLEAR (C) signal from the host.
Bit 14	Watch Dog Timer Violation	Indicates entry into Hyperspace code execution because of a failure to service the Watch Dog Timer.
Bit 15	Spare	Not used, and must be zero.

DOB *ac,50*

This command lets the microECLIPSE CPU control the host PIO interface flags, the Real Time Clock interrupt flags, and the assertion flags for host backplane signals.

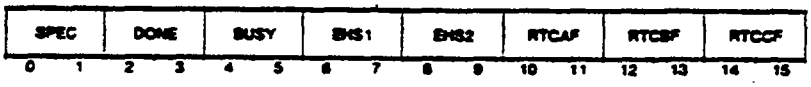
These flags are manipulated by using two bits. The effects of the four possible bit combinations are shown in the following table.

Table 4-1. Interface Flag Bits

1st Bit	2nd Bit	Effect
1	1	None
1	0	Set Flag
0	1	Clear Flag
0	0	Undefined

By using pairs of bits, each flag can be manipulated without affecting the others. For example, to set the Special flag without affecting any of the others, you would set bits 0 and 1 to one and zero respectively. Then, set the rest of the bits in the register to one.

The bit assignments for this register are as follows.



Bits 0-1	Special	Control the state of the host Special flag.
Bits 2-3	Done flag	Control the state of the host Done flag.
Bits 4-5	Busy flag	Control the state of the host Busy flag.
Bits 6-7	EHS1	(External Host Signal Assertion) Reserved for future use.
Bits 8-9	EHS2	(External Host Signal Assertion) Reserved for future use.
Bits 10-11	RTCAF	Control the state of the RTC A interrupt request flag.
Bits 12-13	RTCBF	Control the state of the RTC B interrupt request flag.
Bits 14-15	RTCCF	Control the state of the RTC C interrupt request flag.

Commands Providing Identification

The following two commands, **DIC** *ac*,50 and **DIB** *ac*,50, are used to identify the unique physical address selected for the ILC. Six octets make up the physical address. The first three of these octets are supplied in Hyperspace PROM as a Data General Corporation identification code. The DGC identification code assigned to the controller is 08-00-1B.

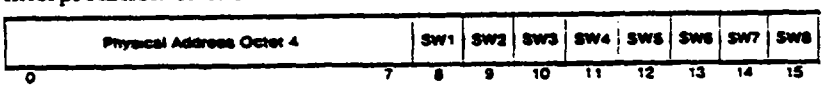
An example of a complete 6-octet unique physical address is:

08-00-1B-E1-00-01

Octet four (E1 in the example) is read from the **DIC** *ac*,50 system control port command. Octet five (00) and octet six (01) are read from the **DIB** *ac*,50 system control port command.

DIC *ac,50*

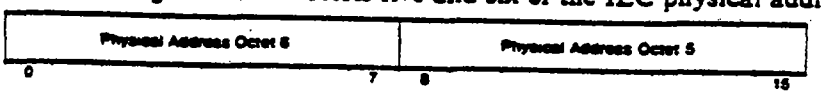
This command, "Read Switch Register," returns the state of 16 DIP switches on the ILC. The interpretation of these switches is as follows.



MSB LSB MSB LSB

Bits 0 - 7	Physical Address Octet Number Four (4)	Reflect octet 4 of the unique 6-octet physical address that has been selected, via switch activation, for the ILC. This is the first of the last three octets transmitted. The order of transmission on the net is from the LSB to the MSB.
Bits 8 - 15	SW1 - SW8	These bits are general purpose identification bits. Their interpretation depends on the resident ILC microECLIPSE system application program.

The following bits reflect octets five and six of the ILC physical address:



MSB LSB MSB LSB

Bits 8 - 15	Physical Address Octet Number Five (5)	Octet 5 of the 6-octet ILC Physical Address. The second of the last three octets transmitted. The order of transmission on the network is from LSB to MSB.
Bits 0 - 7	Physical Address Octet Number Six (6)	Octet 6 of the 6-octet ILC Physical Address. The last octet transmitted on the network. The order of transmission on the network is from LSB to MSB.

Real Time Clock

The Real Time Clock is controlled by System Control Port programming. The clock generates three timing intervals: approximately 5.2 milliseconds (ms), 52 ms, and 520 ms.

The status of the clock intervals is shown in **DIA** *ac*,50 bits 1, 2, and 3 (ARQ, BRQ, and CRQ). Interrupts are masked using **DOA** *ac*,50 bits 1, 2, and 3 (MCA, MCB, and MCC).

The program can detect an interval in one of the following two ways:

1. Monitor the state of the SCP flags (ARQ, BRQ, and CRQ). This method requires continuous polling.
2. Use an interrupt-mask enable for the interval required. When the selected interval expires, this causes an interrupt to the microECLIPSE system.

   The interrupt-mask enables are reset when the system powers up. Enables are controlled with the SCP **DOA** *ac*,50 command bits 1, 2, and 3 (MCA, MCB, and MCC).

   See Appendix B "Device Code Assignments," for the interrupting device codes of the three clock intervals.

The RTC facility is enabled by System Control Port programming using the **DOA** *ac*,50 command bit 15 (CKN). This SCP command enables all three clock intervals to begin setting the ARQ, BRQ, and CRQ flags as each interval times out. This enable command is asynchronous to the RTC device system timing and, therefore, the first setting of the ARQ, BRQ or CRQ flags might not represent proper time intervals. Each succeeding setting correctly reflects the approximate time intervals.

After each interval expiration, the interval flags must be cleared with the SCP **DOB** *ac*,50 command. If the flags are not cleared after a given interval, the RTC cannot indicate the expiration of the next interval.

The host resets the interval flags at system powerup, by issuing an IORST or CLEAR (C) signal. The microECLIPSE system resets the interval flags by issuing an AIORST.

Watch Dog Timer

The ILC controller supports a hardware Watch Dog Timer (WDT) facility that can be used to detect microECLIPSE system code execution abnormalities. When the WDT detects a failure in microECLIPSE system program execution, it issues a WDT NMI. This forces the microECLIPSE system to enter Hyperspace (see Chapter 8, "System Initialization" for an explanation of entry into Hyperspace.)

Once the microECLIPSE system has entered Hyperspace, system functions are controlled by programs in the Hyperspace PROM. For example, the microECLIPSE system can execute a WDT cause handler program from Hyperspace PROM that loads the current Program Counter (before the WDT failure) in the host VIO area, and causes a host interrupt.

When the Hyperspace PROM program determines that the source of the NMI was the WDT it does the following:

1. Disables the WDT facility.
2. Looks for a custom cause handler routine in Hyperspace PROM.

   The method used to find this routine is explained in the section titled *Cause Handler Routines* in Chapter 8, "System Initialization."
3. If the cause handler routine does not exist, the ILC continues to wait for an indication of host serial communication, or host-initiated program load requests.

WDT Operation

The WDT hardware facility is enabled by assertion of bit 14, using the SCP **DOA** *ac*,50 command. Once enabled, the WDT must be serviced. The WDT cannot be disabled by System Control Port programming.

When the WDT is enabled, a service time out begins. Failure to clear **DOA** *ac*,50 bit 14 within one service time out interval results in a hardware-forced WDT NMI to the microECLIPSE processor, and the jump to Hyperspace.

The service time out is synchronized with the slowest RTC interrupt interval (520 ms). If the microECLIPSE system has the RTC facility enabled, it can use the first RTC interrupt after WDT is enabled as a marker for the begining of the service time out.

A service time out is 16 $\pm$ 1 (520 ms) RTC intervals, or approximately 8 seconds. The microECLIPSE system has 16 $\pm$ 1 RTC intervals in which to clear **DOA** *ac*,50 bit 14; if it fails to do so, an NMI is issued. Once bit 14 has been cleared, a new 8-second service time out interval begins and the microECLIPSE system must clear bit 14 again. This continues until the microECLIPSE system fails to clear the bit quickly enough, or until the WDT is disabled. The WDT can be disabled by any of the following:

- System powerup
- A host-issued CLEAR or IORST command
- A write to any address in the upper 16 pages of ILC Hyperspace memory (77777_8-40000_8). (The ILC microECLIPSE system must be executing from Hyperspace in order to write to Hyperspace memory.)

Disabling the RTC facility does not disable the WDT function. If the RTC facility is disabled, timing for the WDT service interval is the responsibility of the microECLIPSE system.

Chapter 5
Data Mover Facility

Chapter 1 briefly described the two-step transfer of data between the host and ILC memory bank B by the Data Mover, and between memory bank B and the Ethernet by the Intel 82586 Local Area Network Communications Controller (LANCC). The Data Mover and the LANCC are both part of the Link system. This chapter describes the functioning and programming of the Data Mover.

ILC local memory bank B is shared by the microECLIPSE system and the Link system (Intel 82586 LANCC and Data Mover). Both the microECLIPSE and Link systems have read/write access to all of memory bank B.

The Data Mover is programmed by the microECLIPSE CPU. Some commands are issued to the Data Mover directly, while others are issued via the System Control Port.

Once programmed, the Data Mover supervises the transfer of data between memory bank B and the host, letting the microECLIPSE system perform other, higher-level functions.

The Data Mover data channel facilities and microECLIPSE system data channel facilities are logically independent. Simultaneous data channel requests from the microECLIPSE system and the Data Mover are not allowed. If the Data Mover and the microECLIPSE system both request access to the data channel at the same time, the data channel honors the first request that it receives, and then alternates channel cycles.

Data transfer is accomplished by the Data Mover in one direction at a time, resulting in a byte swap of data words. Byte swapping is done automatically by the hardware, in contrast to data transfers facilitated by the microECLIPSE system; microECLIPSE system transfers are not byte swapped by hardware.

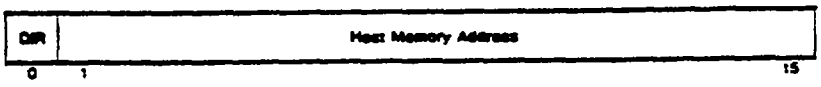
Data Mover Programming

The device code for the Data Mover is 30. The following commands are issued by the microECLIPSE system directly to the Data Mover. The mnemonic MVR stands for the device code 30.

DOA *ac,MVR*

This command specifies the address in host memory to which a block of data is to be moved, or from which a block of data is to be moved, and the direction of the transfer.

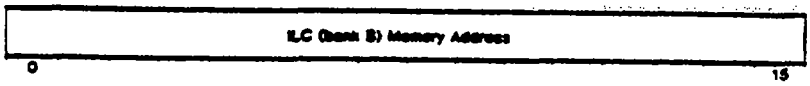
This address is mapped via the host data channel mapping facility using the host map specified in the **DOC** *ac*,MVR command. The microECLIPSE ststem local mapping unit is not used durring Data Mover data channel transfers.



Bit 0	Direction	Specifies the direction of data transfer. SET(1): Data moves from host memory to ILC memory bank B. CLEAR(0): Data moves from ILC memory bank B to host memory. This bit is latched, and remains in the same state until it is reset. Multiple transfers of data in the same direction can be made without reprogramming this bit. There is no default state for this bit. It must be programmed before data is transferred.
Bits 1 - 15	Host Memory Address	These bits are the low-order 15 bits of the host memory address. They point to an address in host memory. For transfers of data from the host to the controller, this address specifies the start of the host memory location to be read. For transfers of data from the controller to the host, this address specifies the start of the host memory location to be written to. These bits are loaded directly into an address counter, and increment as each word transfer takes place.

DOB *ac,MVR*

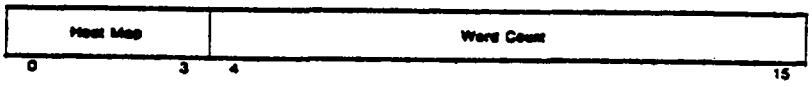
This command is used to load the local ILC memory (bank B) address of the first word of the transfer block.



Bits 0 - 15	ILC Memory Address	Specify the address of the first word of the transfer block in ILC memory (memory bank B). For transfers of data from the host to the controller, this address specifies the start of the bank B memory area to be written to. For transfers of data from the controller to the host, this address specifies the start of the bank B memory area to be read from. These bits are loaded directly into an address counter, and increment as each word transfer takes place.

DOC *ac,MVR*

This command loads the identification number of the data channel map to be referenced, and the number of words to be transferred.



Bits 0 - 3	Host Map	Specify which one of sixteen possible host data channel maps should be referenced during the data transfer. The default state selects map A. These bits are latched, and do not change unless they are reprogrammed.
Bits 4 - 15	Word Count	Specify the number of words in a transfer. Block transfers of up to 4095 words can be specified. If you specify a word count of zero, no data transfer occurs, and the Data Mover Done flag (**DIA** *ac*,50; bit 7) is not set. Do not specify transfers beyond page bounderies, unless contiguous map slots have been assigned and are properly programmed.

The following Data Mover commands are issued via the System Control Port (device code 50).

MVR START (DOA *ac,50; bit 7)*

This command begins block movement. It should be issued only *after* the block parameters have been set up using the Data Mover programming commands, **DOA** *ac*,MVR; **DOB** *ac*,MVR; and **DOC** *ac*,MVR.

The **MVR START** command is issued by setting bit 7 in the System Control Port DOA register (bit 7 = 1). For more information, see Chapter 3, "System Control Port Programming."

MVR CLEAR (DOA *ac,50; bit 8)*

This command clears **MVR DONE**.

The **MVR CLEAR** command is issued by setting bit 8 in the System Control Port DOA register (bit 8 = 1). For more information, see Chapter 3, "System Control Port Programming."

MVR DONE (DIA *ac,50; bit 7)*

This flag is set when the Data Mover completes a block transfer. An interrupt is generated after the MVR DONE flag is set, provided that interrupts are enabled and the Data Mover has not been masked.

You can mask Data Mover interrupts using System Control Port programming. Bit 4 of the **DOA** *ac*,50 System Control Port command controls the Data Mover interrupt mask. Bit 4 of the **DIA** *ac*,50 System Control Port command reports Data Mover mask status.

The interrupt service routine receives MVR interrupt (device 57) when it issues the **INTA** instruction, but no status information is available. The Data Mover does not report the reason for the interrupt.

The microECLIPSE system can monitor the Data Mover transfers. This is done after the **MVR START** signal has been sent, and is accomplished by monitoring the MVR DONE flag, using the **DIA** System Control Port command.

See Chapter 3, "System Control Port Programming" for further information.

MVR RESET (AIORST)

This command resets the Data Mover Done and Data Channel Request flags.

The MVR word counter, ILC memory (bank B) address counter, and host address counter are not initialized by the **AIORST** command. These counters have no defined default state; you must program them before controlled block transfer can occur.

Chapter 6
microECLIPSE Processor/Intel 82586 Interface

The Intel 82586 Local Area Network Communications Controller (LANCC) supervises the transfer of data between an Ethernet/IEEE 802 and memory bank B of the Intelligent LAN Controller (ILC).

This chapter gives a general description of the 82586 LANCC and describes the programming interface between it and the microECLIPSE CPU. You can find detailed information about the 82586 LANCC in the Intel Corporation *82586 Reference Manual*, January, 1983 (order number: 210891-001).

The Intel 82586 LANCC has two internal processors: the Command Unit (CU) and the Receive Unit (RU). Each unit accepts commands from the microECLIPSE CPU.

The microECLIPSE CPU and the LANCC share all of memory bank B. This provides a communications link between the two devices. In addition to this memory-managed interface, the microECLIPSE CPU is able to issue Programmed Input/Output (PIO) commands to the LANCC via the System Control Port.

The memory-managed interface and the PIO commands are described in the next two sections.

We have tried to make our descriptions match those in the Intel manual. For example, we use the same naming conventions. You should, however, *note these differences:*

- Data General Corporation numbers bit diagrams in ascending order, from left to right. Intel numbers bit diagrams in descending order from left to right. In both cases, bit diagrams read from left to right. The most significant bits are on the left and the least significant bits are on the right. *We have adopted Intel's numbering scheme for this chapter only.*.
- We use hash marks (///) to indicate fields that contain reserved or unused data. Intel uses blanks, hash marks, or zeros to mark such fields.
- The command blocks in this chapter are given in sequential word order. Word zero is given first, word one follows, and so on.
- Intel numbers words according to their offset byte, using the byte address of the least significant byte of the word. Divide this address by 2 to obtain the word address. For example, byte address 352_8 is word address 165_8.

Memory Managed Interface

The following data structures have been defined for the Intel 82586 LANCC chip:

- System Control Block
- Command Block
- Transmit Buffer Descriptor
- Receive Frame Descriptor
- Receive Buffer Descriptor.

These data structures reside in memory bank B of the ILC. The microECLIPSE CPU issues control commands to the CU and RU by writing to the System Control Block. The CU and RU then manage transmission and reception of data.

The Command Block and Transmit Buffer Descriptor are part of the Command Unit. The Receive Frame Descriptor and Receive Buffer Descriptor are part of the Receive Unit. The System Control Block, Command Unit, and Receive Unit are described in the following sections.

System Control Block (SCB)

The System Control Block is an eight-word block of RAM. It may be anywhere in ILC memory bank B, as long as the eight words are adjacent.

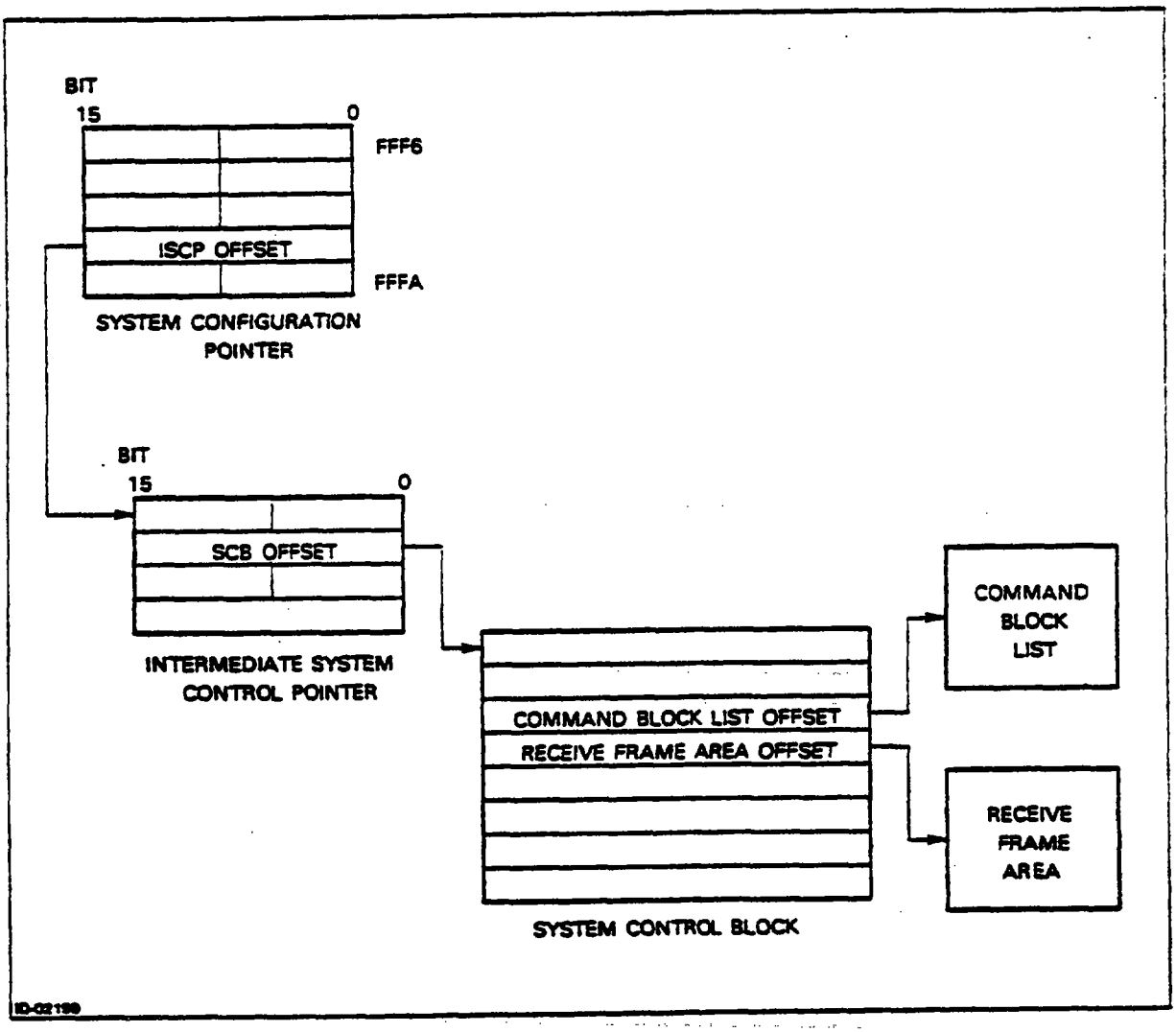


Figure 6-1. Intel Shared Memory Structure

The SCB is pointed to indirectly. Figure 6.1 shows the memory structure in ILC memory bank B that is shared by the microECLIPSE system and the LANCC. The top of Figure 6.1 shows the System Configuration Pointer. The System Configuration Pointer is a block of memory that starts at $FFF6_{16}$. This is the only fixed address data structure in the 82586 system. Part of the System Configuration Pointer is the Intermediate System Control Pointer (ISCP) offset. The ISCP offset starts at address $FFFC_{16}$, and points to the address of the ISCP. The ISCP contains an area called the SCB offset that points to the SCB.

The microECLIPSE CPU issues control commands to the LANCC by writing to the System Control Block, and sending a Channel Attention (CA) signal. The CA signal is sent using the **LNK START** PIO command described later in this chapter.

microECLIPSE CPU control commands have the following functions:

- Controlling the Command Unit
- Controlling the Receive Unit
- Acknowledging an event that caused an interrupt
- Resetting the 82586 LANCC.

The 82586 LANCC writes to the SCB to report the following types of status information to the microECLIPSE CPU:

- Description of the event causing the current interrupt
- Status of the Command Unit
- Status of the Receive Unit
- Statistics related to corrupted receive-frames that have been collected by the 82586 LANCC.

The bit assignments for the eight words of the System Control Block are shown in Figure 6.2. Brief definitions of the bits follow. You can find more detailed information in Intel's *82586 Reference Manual*.

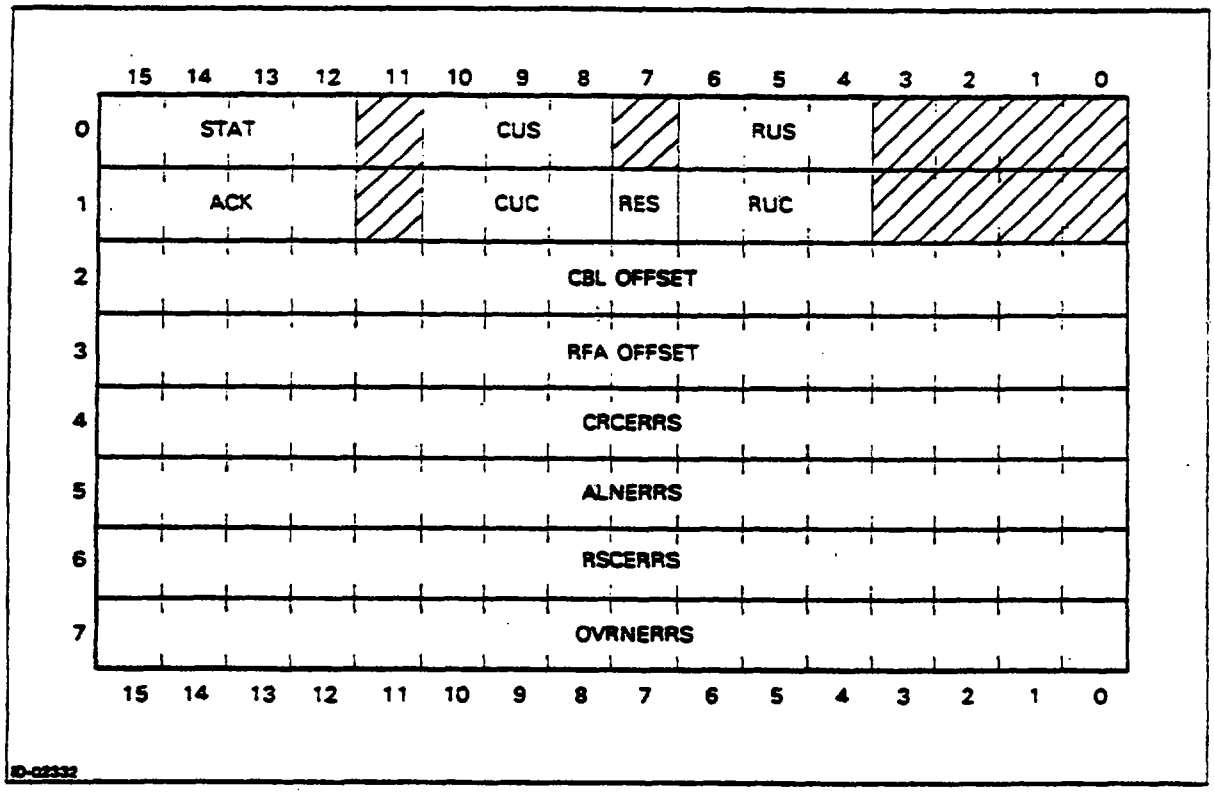


Figure 6-2. System Control Block

Word 0 (Status)

Word 0 shows the status of the Intel 82586 LANCC. These sixteen bits are set and cleared by the LANCC. The microECLIPSE CPU does not modify these bits. Word 0 contains three kinds of status information: STAT (Status), CUS (Command Unit Status), and RUS (Receive Unit Status). The definitions of the individual bits of each of these three parts of word 0 are listed below.

STAT

Bit 15	CX	SET(1): A command in the Command Block List that has its Interrupt (I) bit set has been executed.
Bit 14	FR	SET(1): A frame has been received.
Bit 13	CNR	SET(1): The Command Unit is not ready.
Bit 12	RNR	SET(1): The Receive Unit is not ready.
Bit 11		Not used, and must be zero.

CUS

Bits 10-8 These three bits can be set in seven combinations. The interpretations of the seven possible combinations are as follows:

0_8 Idle
1_8 Suspended
2_8 Ready
$3\text{-}7_8$ Not used.

Bit 7 Not used, and must be zero.

RUS

Bits 6-4 These three bits can be set in seven combinations. The interpretations of the seven possible combinations are as follows:

0_8 Idle
1_8 Suspended
2_8 No resources
3_8 Not used
4_8 Ready
$5\text{-}7_8$ Not used.

Bits 3-0 Not used, and must be zero.

Word 1 (Command)

Word 1 is a command word that specifies the action to be performed when a CA signal is received. This word controls three types of action: ACK (Acknowledge), CUC (Command Unit Command), and RUC (Receive Unit Command). These sixteen bits are set by the microECLIPSE CPU, and cleared by the Intel 82586 LANCC. The individual bits are described below.

ACK

Bit 15	ACK-CX	SET(1): Acknowledges the completion of an action command.
Bit 14	ACK-FR	SET(1): Acknowledges the receipt of a frame.
Bit 13	ACK-CNR	SET(1): Acknowledges that the Command Unit is not ready.
Bit 12	ACK-RNR	SET(1): Acknowledges that the Receive Unit is not ready.
Bit 11		Not used, and must be zero.

CUC

Bits 10-8 These three bits can be set in seven combinations. The interpretations of the seven possible combinations are as follows:

0_8 NOP. Take no action.

1_8 Start execution of the first command in the Command Block List (CBL). If a command is currently in execution, complete it before executing the new command.

The beginning of the CBL is in the CBL OFFSET.

2_8 Resume the operation of the Command Unit (CU) by executing the next command.

This operation assumes that the CU has been previously suspended.

3_8 Suspend execution of commands on the CBL after the current command is complete.

4_8 Abort current command immediately.

$5\text{-}7_8$ Not used, and have no effect.

Bit 7 Reset Resets the chip. This is logically the same as a hardware reset.

RUC

Bits 6-4 These three bits can be set in seven combinations. The interpretation of the seven possible combinations follows.

0_8 NOP. Take no action.

1_8 Start receiving frames. If a frame is currently being received, complete that reception before starting.

The beginning of the Receive Frame Area (RFA) is in the RFA OFFSET.

2_8 Resume receiving frames.

This operation assumes that the RU has been previously suspended.

3_8 Suspend receiving frames. If a frame is currently being received, complete the reception before suspending.

4_8 Abort frame reception immediately.

$5\text{-}7_8$ Not used, and have no effect.

Word 2 (CBL OFFSET)

Bits 15-0 These 16 bits specify the address offset for the first command block in the CBL. The CBL OFFSET is accessed only if the Start command is specified for the Command Unit Command (CUC = 001_2).

Word 3 (RFA OFFSET)

Bits 15-0 These bits specify the address offset for the Receive Frame Area (RFA). This field is accessed only if the Start command is specified for the Receive Unit Command (RUC = 001_2).

Word 4 (CRCERRS)

Bits 15-0 This counter contains the number of aligned frames discarded because of a Cyclic Redundancy Check (CRC) error. The counter is updated when necessary, no matter what the state of the Receive Unit.

Word 5 (ALNERRS)

Bits 15-0 This counter contains the number of misaligned frames discarded because of a CRC error. The counter is updated when necessary, no matter what the state of the Receive Unit.

Word 6 (RSCERRS)

Bits 15-0 This counter contains the number of good frames that were discarded because there were no resources available. This counter is updated only if the Receive Unit is in the No Resources state (RUS = 010_2).

Word 7 (OVRNERRS)

Bits 15-0 This counter indicates how many frames were lost because the local system bus was not available. If the problem lasts for more than the duration of one frame, the frames that follow the first one are lost without an indicator, and are not counted. This counter is updated no matter what the state of the Receive Unit.

Command Unit

Commands in the System Control Block control how the Command Unit executes action commands that are located in the Command Blocks (CB). Command Blocks are linked together to form the Command Block List (CBL). The Command Block List must contain at least one Command Block.

The "transmit" command in the Command Block accesses user data in buffers for transmission operations. Each buffer is described by a Transmit Buffer Descriptor.

The Command Block and Transmit Buffer Descriptor are described in the next two sections.

Command Block

Figure 6.3 shows a Command Block. This block provides header information for every action command. The controller must be able to find this block when it executes an action command. You pass the address of a CBL in the System Command Block (CBL offset).

The Command Block LINK FIELD (word two) points to the next CB in a list.

The last CB in the Command Block List is indicated by setting the End List (EL) bit in word one of the CB. The Command Unit starts at the beginning of the CBL, and continues executing commands, one at a time, until it encounters a CB with EL set to 1.

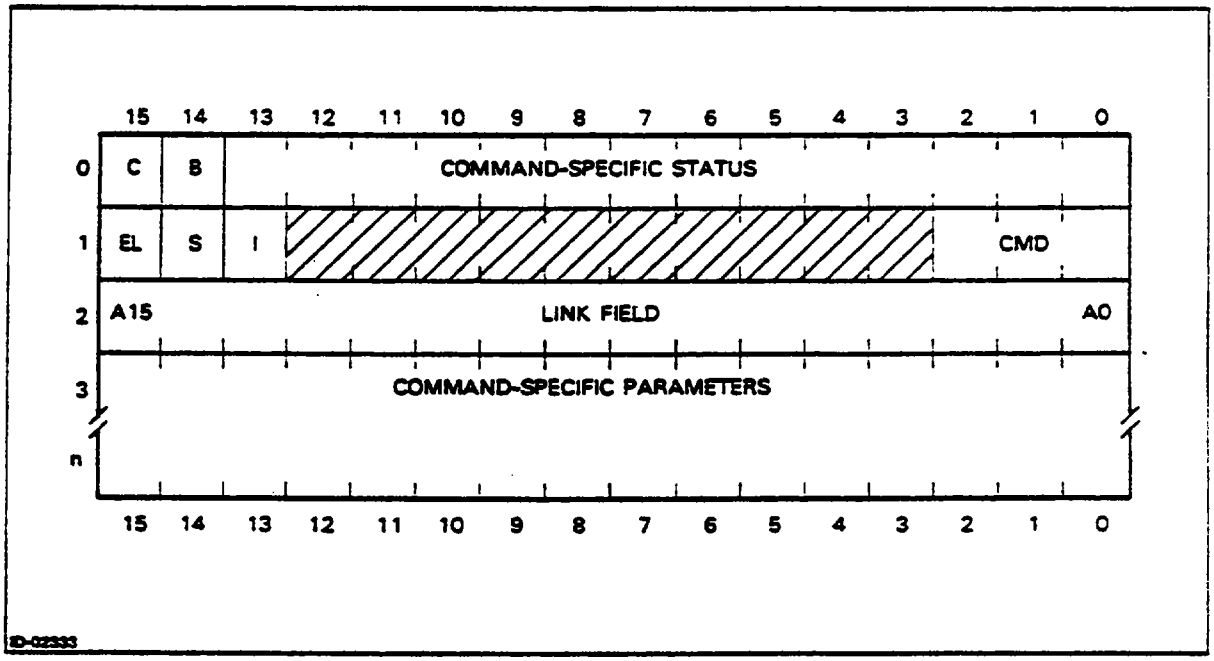


Figure 6-3. Command Block

Word 0 (Status)

Bit 15	C	SET(1): Indicates the completion of execution of an action command. This bit is cleared by the microECLIPSE CPU when the Command Block is placed on the CBL. When command execution is complete the 82586 LANCC sets this bit.
Bit 14	B	SET(1): Indicates that the 82586 LANCC is executing the command specified in word 1, bits 2-0 (CMD). This bit is initially set to zero by the microECLIPSE CPU. When command execution begins, the 82586 sets the bit to one. When command execution is completed the 82586 clears the bit.
Bits 13-0	Command- specific Status	Indicate the results of a command. This field is valid only if the C bit is set. Contents of the field depend on the command, except for bits 13 and 12: Bit 13 SET(1): Indicates that the command executed without error. Bit 12 SET(1): Indicates that the command was aborted with a CU Abort Control Command.

Word 1 (Command)

Bit 15	EL	SET(1): Indicates that this Command Block is the last one in a CBL.
Bit 14	S	SET(1): Suspends the CU when the current Command Block completes.
Bit 13	I	SET(1): Causes the 82586 LANCC to generate an interrupt after execution of the current command. The CX bit in the System Control Block (SCB) is set when this bit is set. CLEAR(0): No interrupt is generated. The CX bit in the SCB is cleared when this bit is cleared.
Bits 12-3		Not used, and must be zero.
Bits 2-0	CMD	Specify the action command code: These three bits can be set in seven combinations. The interpretation of the seven possible combinations follows. 0_8 NOP 1_8 Individual Address Set Up 2_8 Configure 3_8 Multicast Address Set Up 4_8 Transmit 5_8 Reflectometer test 6_8 Dump Status 7_8 Diagnose.

Word 2 (LINK FIELD)

Bits 15-0 These bits point to the next CB.

Words 3-n (COMMAND SPECIFIC PERAMETERS)

This is a variable length field. Its contents depend on action commands. The contents reflect the parameters for, and return values associated with, an action command. See Chapter 4 of the Intel *82586 Reference Manual* for details.

Transmit Buffer Descriptor (TBD)

When the Command Unit issues a Transmit action command, Transmit Buffer Descriptors are used to describe buffers containing user data. Each transmit command may specify zero or more TBDs. The TBDs are linked to form a frame. They are automatically prefetched by the Command Unit as required. Figure 6.4 shows a Transmit Buffer Descriptor (TBD).

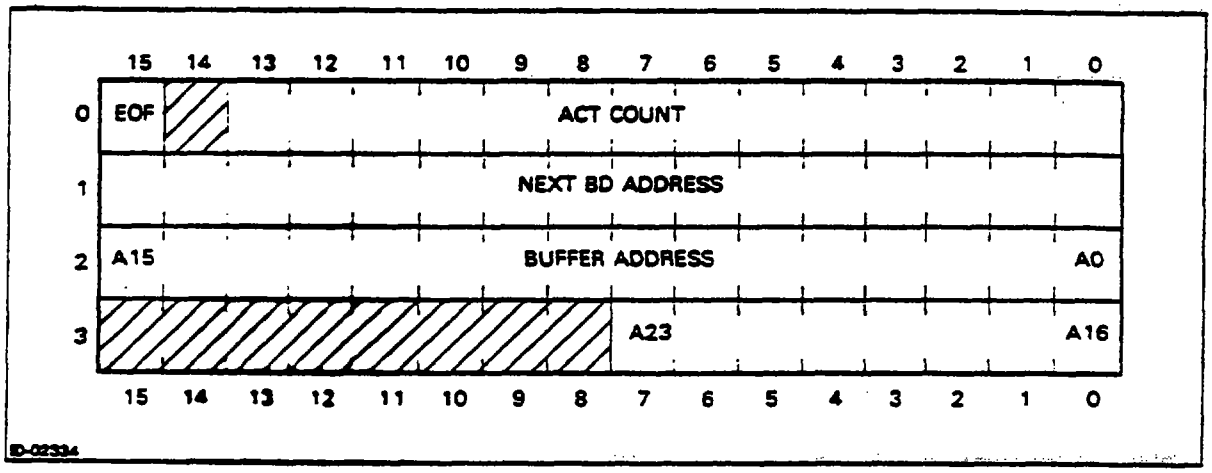


Figure 6-4. Transmit Buffer Descriptor

Word 0

Bit 15	End Of Frame	SET(1): Indicates that the current TBD is the last one in the current frame to transmit. This bit is set by the microECLIPSE CPU before transmission.
Bit 14		Not used, and must be zero.
Bits 13-0	ACT COUNT	Specifiy the number of bytes that contain information for the current buffer. These bits are set by the microECLIPSE CPU before transmission.

Word 1 (NEXT BD ADDRESS)

Bits 15-0 These bits make up the offset portion of the address of the next TBD in the list. This field is meaningless if the EOF bit is set to 1.

Word 2 (BUFFER ADDRESS)

Bits 15-0 The two least significant bytes of the starting address in memory that contains the data to be sent.

Word 3 (BUFFER ADDRESS)

Bits 15-8 Not used, and must be zero.

Bits 7-0 The most significant byte of the starting address of the data to be sent.

Receive Unit

The Receive Unit manages all activities related to frame reception. The Receive Unit is independent of the Command Unit, except that the Receive Unit uses the Command Unit to communicate with the microECLIPSE CPU.

The Receive Unit controls a memory area called the Receive Frame Area (shown in Figure 6.5). The Receive Frame Area contains two lists, the Received Frame List and the Free Frame List.

The Received Frame List contains valid data, while the Free Frame List contains empty buffers. The last frame of the Receive Frame List points to the Free Frame List.

The Free Frame List is composed of two lists:

Receive Descriptor List This is a list of empty Receive Frame Descriptors.

Free Buffer List This is a list of empty buffers. Each free buffer is described by a Receive Buffer Descriptor.

The address of the Receive Frame Area is given to the Receive Unit by the microECLIPSE CPU, using the System Control Block and a START signal; this is the address of the first Receive Frame Descriptor on the Receive Descriptor List.

One Receive Frame Descriptor is used for each frame received. Any number of Receive Buffer Descriptors can be used. The number of RBDs is determined by how many buffers are required to contain the frame.

If either the Receive Descriptor List, or the Free Buffer List is exhausted, the Receive Unit notifies the microECLIPSE CPU, and enters the No Resources state. This means that there is no way to handle additional Receive Frames.

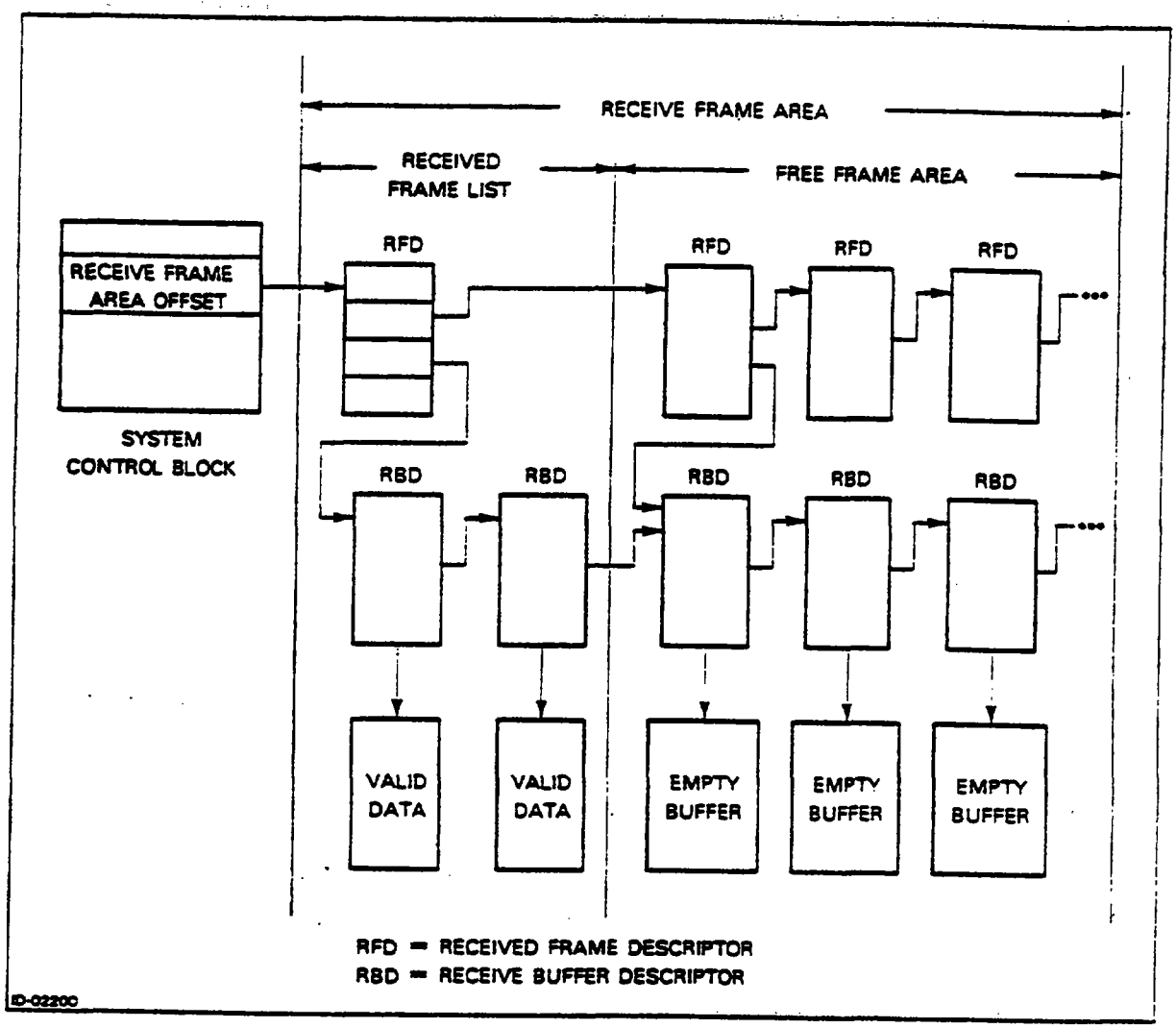


Figure 6-5. Receive Frame Area

Receive Frame Descriptor

Each frame that is received is described by one Receive Frame Descriptor (RFD). Figure 6.6 shows a RFD. Individual bits are defined below. For more detailed information, see the Intel *82586 Reference Manual.*

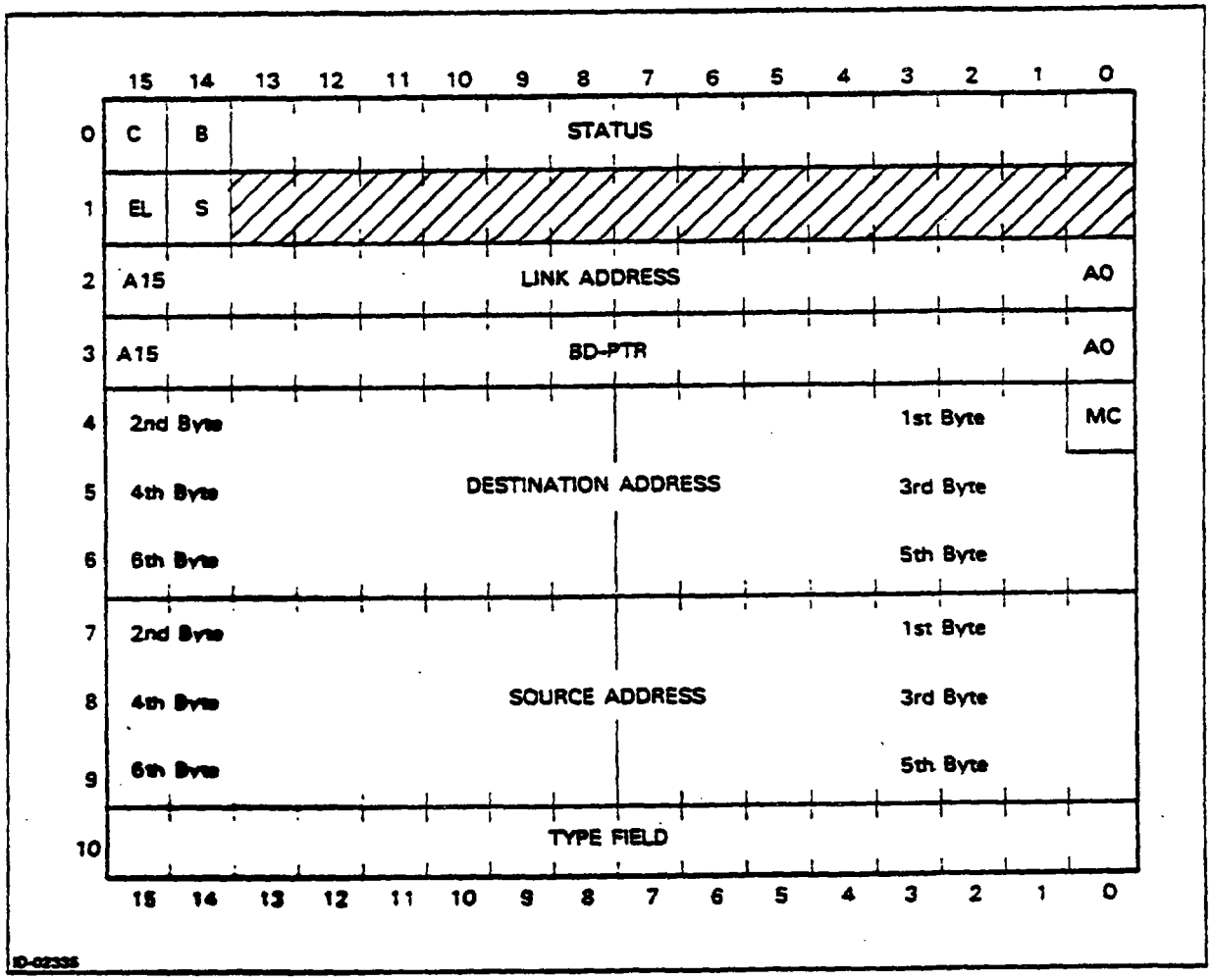


Figure 6-6. Receive Frame Descriptor

Word 0 (STATUS)

Bit 15	C	SET(1): Indicates completion of execution of an action command. This bit is set by the 82586 LANCC.
Bit 14	B	SET(1): Indicates that the 82586 is ready to receive or in the process of receiving this frame. This bit is initially cleared by the microECLIPSE CPU. When reception setup begins, the 82586 sets the bit. When reception is complete the 82586 clears the bit.
Bit 13		SET(1): Indicates that a frame was received without errors.
Bit 12		Not used, and must be zero.
Bit 11		SET(1): Indicates a CRC error in an aligned frame.
Bit 10		SET(1): Indicates an alignment error.
Bit 9		SET(1): Indicates that there is no more buffer space available.
Bit 8		SET(1): Indicates a DMA overrun.
Bit 7		SET(1): Indicates that the frame is too short.
Bit 6		SET(1): Indicates no End of File flag (for bitstuffing only).
Bits 5-0		Not used, and must be zero.

Word 1 (EL, S)

Bit 15	EL	SET(1): Indicates that this Receive Frame Descriptor is the last one in a Receive Descriptor List.
Bit 14	S	SET(1): Indicates that the Receive Unit will suspend after receiving the current frame.
Bits 13-0		Not used, and must be zero.

Word 2 (LINK ADDRESS)

Bits 15-0 These bits point to the next Receive Frame Descriptor. The Link Address of the last frame can be used to form a cyclic list.

Word 3 (BD-PTR)

Bits 15-0 This is the offset portion of the address of the first Receive Buffer Descriptor (RBD) containing frame data. If this field contains $0FFFF_{16}$, there is no RBD.

Words 4-6 (DESTINATION ADDRESS)

Bits 15-0 This field contains the destination address of the current receive frame. The field is 0 to 6 bytes long.

Words 7-9 (SOURCE ADDRESS)

Bits 15-0 This field contains the source address of the current receive frame. The field is 0 to 6 bytes long.

Word 10 (TYPE FIELD)

Bits 15-0 These bits specify the type field of the current receive frame. The type field is 2 bytes long.

Receive Buffer Descriptor (RBD)

The Receive Buffer Descriptor is pointed to by the Receive Frame Descriptor. Figure 6.7 shows a Receive Buffer Descriptor (RBD). This block in turn defines the location of user data in the controller. This is data that has been received by the 82586 LANCC from a source other than the microECLIPSE CPU.

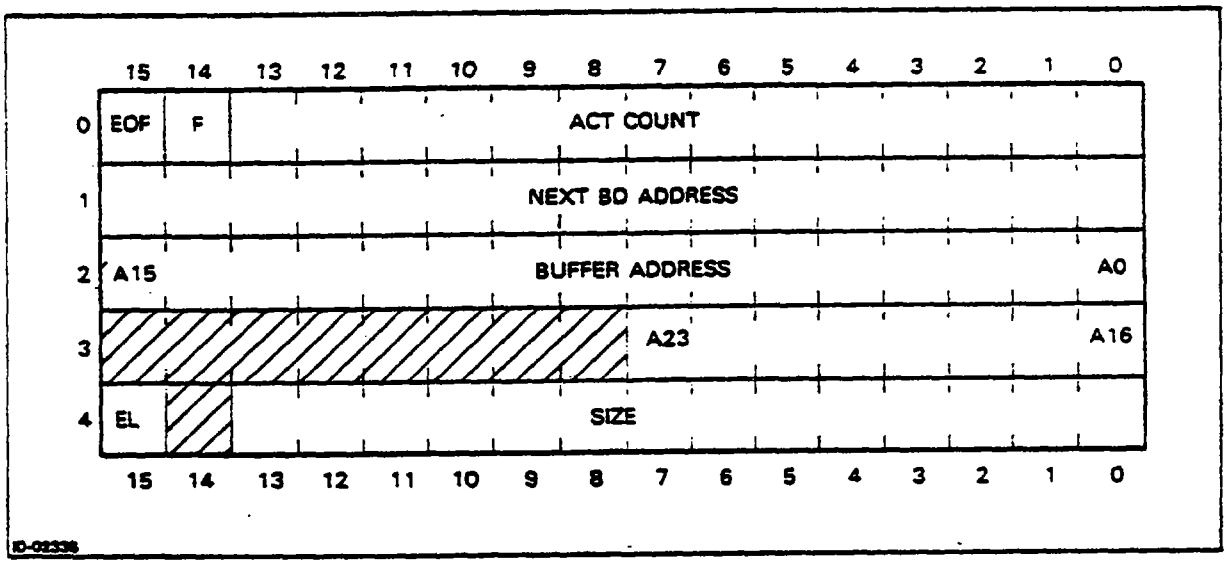


Figure 6-7. Receive Buffer Descriptor

Word 0 (EOF, F, ACT COUNT)

Bit 15	End of Frame	SET(1): Indicates that the current RBD is the last one in the current frame. This bit is cleared by the microECLIPSE CPU before starting the Receive Unit. It is set by the 82586 LANCC after reception of the frame.
Bit 14	F	SET(1): Indicates that the current buffer has already been used. This bit is cleared by the microECLIPSE CPU before starting the Receive Unit, and set by the 82586 LANCC after the buffer has been used.
Bits 13-0	ACT COUNT	SET(1): Specify the number of meaningful bytes in the buffer. This bit is cleared by the microECLIPSE CPU before starting the Receive Unit, and set by the 82586 LANCC after the buffer has been used. This field is meaningless if the F bit is clear(0). If ACT COUNT is odd (in word mode), the 82586 writes garbage to the high byte of the last word.

Word 1 (NEXT BD ADDRESS)

Bits 15-0 These bits form the offset portion of the address of the next RBD in the list. This field is meaningless if EOF is set(1).

Word 2 (BUFFER ADDRESS)

Bits 1-15 These bits make up the two least significant bytes of the starting address of the area in memory that contains the received data.

Word 3 (BUFFER ADDRESS)

Bits 15-8 Not used, and must be zero.

Bits 7-0 These bits make up the most significant byte of the starting address of the data that has been received.

NOTE: In word mode, the buffer address must be even.

Word 4 (EL, SIZE)

Bit 15	EL	SET(1): Indicates that the associated buffer is the last one on the current Free Buffer List.
Bit 14		Not used, and must be zero.
Bits 13-0	SIZE	Specify the number of bytes in the associated buffer. In word mode, this quantity must be even.

Programmable Input/Output (PIO) Commands

PIO commands are sent to the 82586 LANCC via the System Control Port. The functional handshake between the microECLIPSE CPU and the LANCC consists of a Channel Attention (CA) signal sent by the microECLIPSE CPU to the LANCC, and an interrupt signal sent by the LANCC to the microECLIPSE CPU.

The CA signal is caused by the **LNK START** command. The interrupt is caused by the **LNK DONE** command.

Information about the System Control Port can be found in Chapter 4. Detailed information about the microECLIPSE CPU/LANCC functional handshake can be found in the Intel *82586 Reference Manual.*

LNK START (DOA *ac,50; Bit 12)*

The microECLIPSE CPU first sets up the Memory Managed control structures in ILC memory bank B. These control structures include commands to be executed by the LANCC. The commands are either defined or indexed in the System Control Block.

The microECLIPSE CPU causes the LANCC to execute a command by sending a Channel Attention (CA) signal. A CA can be sent by issuing the **LNK START** command.

The LANCC latches the falling edge of the CA signal and begins execution of the command. When the command is executed, the LANCC clears the command word in the System Control Block to zeros.

The LANCC will not accept a second CA signal until the command word in the SCB has been cleared to zeros. **LNK START** commands issued before the command control word has been cleared by the LANCC may be lost.

LNK CLEAR (DOA *ac,50; Bit 9)*

This command clears the Link Done flag. The LANCC sets its Done flag to send an interrupt to the microECLIPSE CPU. The program should first dismiss the interrupt by issuing a **LNK CLEAR**, and then service the interrupt request.

In order to receive an new interrupt request from the LANCC, the microECLIPSE CPU must set the ACK bits in the System Control Block, and send a CA signal to the LANCC. If any interrupts are pending when the CA is issued, the Link Done flag is immediately reset.

LNK DONE (DIA *ac,50; Bit 12)*

This command monitors the 82586 LANCC Done flag. The Done flag is set when one of the following events occurs:

- Execution of a command is complete.

  The command must be in a Command Block pointed to by the System Control Block, and the "I" bit in the Command Block must be set to 1.
- A frame is received.
- The Command Unit becomes not ready.
- The Receive Unit becomes not ready.

LANCC interrupts are masked by setting the "MLK" bit (Bit 5) in the System Control Port **DOA** *ac*,50 command.

LNK RESET (AIORST)

This command initializes the 82586 LANCC, and resets the Done and memory bank B bus request flags.

LNK RESET is at least four Intel 82586 system clock periods in length, and not more than one AIORST instruction time.

Chapter 7
I/O Interface

The I/O interface is provided by a transceiver and a Fujitsu MB502A chip.

Transceiver

The transceiver connects the ILC to the Ethernet/IEEE 802 network. The Ethernet coaxial cable is broken, and the transceiver attached in line with the cable. The total length of all cables between the ILC and the Ethernet is a maximum of 50 meters. DGC supplies either a 5-meter or a 20-meter teflon cable with the ILC.

CSMA/CD

The ILC provides access to the network using the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol. This access method lets the ILC avoid transmitting messages at the same time another node is transmitting. The carrier sensing and collision detection functions are provided by the tranceiver.

The transceiver monitors coaxial cable transmissions to see if any messages are being sent (carrier sensing). If it finds that none are being sent, the transceiver indicates to the Intel 82586 that it may proceed with a transmission.

The transceiver then monitors the cable during transmission, looking for the presence of multiple transmissions (collision detection). Multiple transmissions indicate that a packet collision might have occurred. When multiple transmissions are detected, the transceiver signals the 82586; the 82586 stops transmission and waits a random period of time before initiating transmission again.

Fujitsu MB502A

The Fujitsu MB502A provides the electrical interface to the Ethernet/IEEE 802 transceiver cable, and operates directly with the Intel 82586 LANCC.

The Fujitsu MB502A chip:

- Performs Manchester encoding and decoding of Ethernet/IEEE 802 data
- Contains the differential drivers and receivers for connection to the transceiver
- Passes carrier-sensing and collision-detection information from the transceiver to the 82586 LANCC using TTL level carrier-sense, and collision-detection signals
- Provides transmit and receive clocks for the Intel 82586.

The Fujitsu MB502A requires no direct programming by the microECLIPSE processor, other than the assertion of the LOOPBACK function (see Chapter 4, "System Control Port Programming").

For a detailed account of the device's functions, please reference the Fujitsu MB502A Encoder/Decoder data sheet.

Chapter 8
System Initialization

The ILC can reference separate memory locations, User Space and Hyperspace. System initialization is done by using programs located in Hyperspace memory.

Hyperspace is entered whenever a Non Maskable Interrupt (NMI) is received by the microECLIPSE processor. The NMI causes the microECLIPSE system to execute system initialization programs stored in a PROM located in Hyperspace memory. The programs provided by DGC are self-test and boot programs. There is also space available in the Hyperspace memory for user programming; the user can program routines that respond to the cause of interrupts (cause handler routines), service requests by the host, or perform any other program function.

The first four sections of this chapter describe the ILC User Space and Hyperspace memory locations, how Hyperspace is accessed, the functions of Hyperspace programs, and the sequence of events in system initialization. The final two sections provide you with information needed to add cause handler routines, and to perform ILC program loading of the host.

Memory Locations

User Space memory consists of 128 pages of dynamic RAM located in ILC local memory banks A and B. It encompasses all of memory banks A and B, and is logically divided into System Memory and User Memory.

The 32-page Hyperspace memory, shown in Figure 8.1, is physically separate from User Space memory.

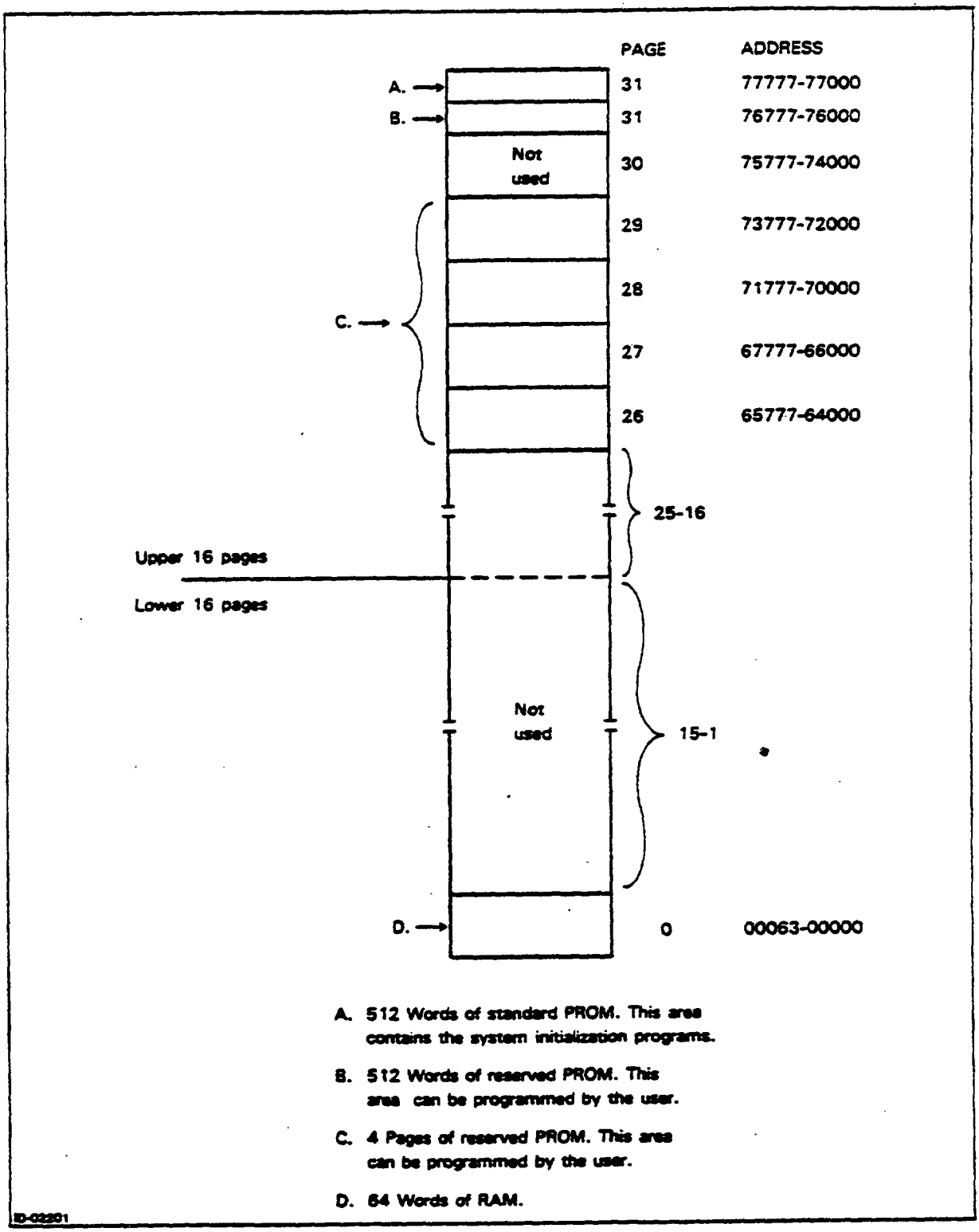


Figure 8-1. Hyperspace Memory Locations

Hyperspace memory includes 512 words of standard PROM that contain the programs used during system initialization. These 512 words are located at the top of Hyperspace at logical address 77777_8-77000_8.

The 512 words located at logical address 76777_8-76000_8 and the four pages of PROM located at logical address 75777_8-66000_8 are available for users to program (custom PROM).

Also, one page (page 0) of Hyperspace RAM contains 64 words that the microECLIPSE system can write to during system initialization or at any other time that the microECLIPSE processor is executing from Hyperspace. The 64 words of RAM are located at address 00000_8-00063_8.

Entering Hyperspace

Communication between the microECLIPSE CPU and the other components of the ILC occurs over a single 16-bit I/O bus. Each bus cycle is divided into two phases.

The first phase of the cycle specifies the type of transaction, and the target of the transaction. The second phase of the cycle transmits or receives data.

The ILC microECLIPSE system can carry out three types of transactions:

- A memory transaction from User Space
- A memory transaction from Hyperspace
- An I/O cycle.

The microECLIPSE processor specifies which type of transaction is taking place by using the first (0) bit of the 16-bit logical address, and the Memcycle pin. If the Memcycle pin is asserted, the transaction is a memory cycle; if it is not asserted, the transaction is an I/O cycle.

When the Memcycle pin is asserted, the first bit of the logical address distinguishes the kind of memory cycle: SET is a Hyperspace memory transaction; CLEAR is a User Space memory transaction. This is shown in Figure 8.2:

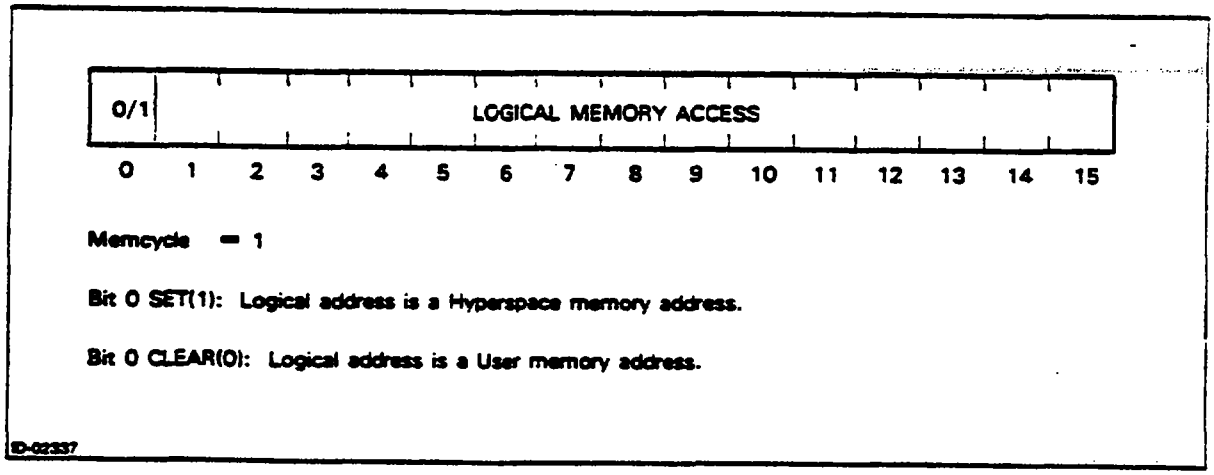


Figure 8-2. Memory Cycles

Receipt of an NMI causes entry into Hyperspace. An NMI is caused by the following:

- IORST or CLEAR signal (programmed reset) from the host
- RESET signal (powerup or console reset) from the host
- Indication of a write violation (**DIA** *ac*,50; bit 13 is SET)
- Indication that the Watch Dog Timer has detected an error (**DIA** *ac*,50; bit 14 is SET)
- Indication of a voluntary entry into Hyperspace (**DIA** *ac*,50; bit 6 is SET).

Hyperspace Program Functions

The execution of the system initialization programs stored in Hyperspace PROM can do the following:

- Perform an automatic PROM checksum test. This test computes the Modulo 65536 sum of all 512 locations in hyperspace PROM. If the sum is equal to zero, the program indicates a pass, and begins execution. If this test fails, the microECLIPSE CPU executes a **JMP .** and hangs at a Hyperspace address that indicates PROM checksum failure.
- Perform a "sanity check" that tests microECLIPSE system registers and ALC instructions to verify basic microECLIPSE CPU operation. The program indicates a pass if CPU operation is acceptable. If the test fails, the microECLIPSE processor executes a **JMP .** and hangs at a Hyperspace address that indicates a sanity check failure.
- Determine the event, or events, that caused the microECLIPSE system to enter Hyperspace, and whether an individual cause handler routine exists for each event. If a cause handler routine does exist, the PROM program finds and jumps to the routine so that the routine can execute.

  If the cause of entry into Hyperspace was a RESET, IORST, or CLEAR signal, the microECLIPSE system indicates a pass, and enters its polling state.
- Detect Program Load requests made by the host, and determine whether a Program Load handler routine exists. If a Program Load handler routine does exist, a PROM program finds the routine and jumps to it so that the routine can execute.
- Determine whether there are user-developed (custom) Hyperspace PROM programs, and if so, find the appropriate custom programs. These may be the cause handler or program load handler routines mentioned earlier, or any other type of program.
- Support protocol control for a half-duplex serial communication link with the host.

  This protocol uses the Busy, Done, and Special flags of the ILC System Control Port. The protocol supports passing of a host address that points to a virtual I/O (VIO) register block. The VIO block is used to implement I/O commands over the data channel. The protocol also supports interpretation of several special I/O commands to help the system user initilize and download the controller from the host.
- Give the user an optional memory-destructive self-test diagnostic that runs on the microECLIPSE system, and 128 pages of available local User Space memory. The self-test provides a visual indication of a successful pass.

Sequence of Events During Initialization

When the ILC undergoes system initialization, the following sequence of events occurs:

1. The event causing initialization sends an NMI to the microECLIPSE processor.
2. The Program Counter is saved in location 00010_8 of Hyperspace RAM.
3. The microECLIPSE processor jumps to location 77777_8 of Hyperspace PROM.

4. The data currently in the microECLIPSE system accumulators is saved by writing the data to Hyperspace RAM.
5. The microECLIPSE system sanity check is performed.
6. The PROM checksum test is performed.
7. The microECLIPSE CPU reads the SCP status register (**DIA** 0, 50), bits 6 (Voluntary Entry), 13 (Write Violation) and 14 (Watch Dog Timer), to determine the cause of the jump to Hyperspace. If one of these bits is SET, the microECLIPSE system:
   - Stores the SCP status information in Hyperspace RAM location 000012_8
   - Issues an AIORST (microECLIPSE system reset) that clears the NMI status
   - Turns off the Watch Dog Timer by performing a write to any location in the upper half of Hyperspace memory (logical address 77777_8-40000_8)
   - Checks for appropriate cause handler routines.

   If none of these bits are SET, the cause of the entry into Hyperspace is interpreted as a RESET, IORST, or CLEAR signal. In this case, the microECLIPSE system:
   - Turns on its diagnostic light (pass indicator)
   - Enters a polling state, and waits for a START or PULSE signal from the host.

     The START sets the controller's Busy flag, indicating that the host wants a program load from the ILC (see the section on *Program Loading* later in this chapter).

     The PULSE sets the controller's Special flag, indicating the beginning of host to ILC serial communication (see Chapter 2, "Host/Controller Programmed Input/Output Interface").

   The ILC microECLIPSE system remains in the polling state until the host intervenes.

Cause Handler Routines

Cause handler routines can exist in either Hyperspace memory or in User Space memory. They are user-programmed routines that can be executed in response to an NMI caused by a write violation, Watch Dog Timer error detection, or voluntary entry into Hyperspace, and in response to a host request for a program load.

The microECLIPSE system first determines whether a specific cause handler routine exists by looking for the key value, 161742_8 at a specific address. The key can be located in Hyperspace memory, or in User Space memory; the specific locations are shown in Table 8.1. The microECLIPSE system always checks for keys in User memory first. If none is found, the microECLIPSE system looks in Hyperspace memory to see if the key exists there.

For example, if the Hyperspace enter was caused by a write violation, the microECLIPSE system retrieves a key pointer value from Hyperspace location 077137_8. The pointer value indicates that the User Space key is at User Space location 000054_8. The microECLIPSE system then reads User Space location 000054_8 and checks to see if this location has the key. If the key is not there, the microECLIPSE system checks for the Hyperspace key.

If the User Space key is in location 000054_8, the microECLIPSE system jumps to the key location plus one. The address of the first instruction of the cause handler routine is written in this location. The microECLIPSE system can go to the address specified in this "key location plus 1," and begin executing the cause handler routine.

If you want to build special user space or Hyperspace cause handler routines, refer to the following table of address data.

Table 8-1. Keys Locating Cause Handler Routines

Function	Hyperspace Key Location	Hyperspace Address Location	User Space Key Location	User Space Address Location
Boot Code	-	-	000050	000051
Enter Hyperspace	-	-	000052	000053
Write Violation	-	-	000054	000055
WDT Violation	-	-	000056	000057
Hyperspace PROM	64000	64001	-	-

Program Loading

Program loading takes place when the microECLIPSE system is in the polling state described in step 7 in the earlier section, *Sequence of Events During System Initialization.*

If the ILC receives a START signal while it is in this polling state, the programs in the Hyperspace PROM access the User Space key location 000050_8 (see the preceding section for an explanation of keys). This is the key location for boot code; the ILC checks to see if this location contains the key.

If the key is there, the microECLIPSE system jumps to the key location plus one. This location contains the address of the microECLIPSE system User Space boot handler routine. If the User Space key is not there, the microECLIPSE processor reads the Hyperspace key location (64000_8, to see if the key is there. If the Hyperspace key is there, the microECLIPSE system jumps to the key location plus one, which contains the Hyperspace address of the custom Hyperspace boot handler routine.

If the microECLIPSE system receives a Host START signal and neither User Space nor Hyperspace boot handling code exists (neither key is found), the microECLIPSE system ignores the START and continues to poll, waiting for a PULSE.

Network Assisted Program Loading

This function is supported by ILC hardware only; to implement it, you must provide your own custom software in Hyperspace PROM.

When a host reset occurs because of an automatic reset after a power failure (power-fail auto start), or because of a console reset, the host can use the ILC and be booted from the network. Booting from the network means that the host would not need to have its own disk; it could request boot packets from any node on the network that is acting as a distributed disk or general file server.

The events that occur during a network-assisted program load of the host are:

1. A reset is caused by power-fail auto restart, or a reset from the operator console.
2. The host CPU performs an automatic program load to a device specified by the user. In this case, the device must be the ILC.
3. A START signal is received by the ILC. The START signal is caused by power-fail auto restart, or by a program load from the operator console.
4. The standard boot program in Hyperspace PROM causes the microECLIPSE system to find a custom program in Hyperspace PROM.
5. The custom program requests boot packets from a network node, and performs a standard high speed channel load of the host processor.

The protocol governing the network boot request and the packet format is defined by the custom software.

Appendix A
Instruction Set

This appendix describes the ILC microECLIPSE instruction set. These instructions are a subset of the ECLIPSE S/20 instruction set. For further details, see the *ECLIPSE® S/20 Assembly Language Programmer's Reference* (014-000682).

Symbols and Terms

The following table lists the symbols and terms that are used in this appendix to define the ILC instructions. Each symbol or term is briefly defined in the table. Those table entries preceded by an asterisk (*) are defined in more detail in the following sections of this appendix.

Table A-1. Symbols and Terms

	Symbol or Term	Description
	[]	Indicate that the enclosed symbol is optional
	N	An integer in the range 0 to 3
	n	An integer in the range 1 to 4
*	@	Indicates whether an address is a pointer to another location or the effective address (indirect bit)
*	f	Control field that manipulates the Busy and Done flags
	AC or ac	Accumulator
	ACS	Source accumulator
	ACD	Destination accumulator
*	c	Inititalizes the carry bit
	#	Do not load the result of an ALC operation into the destination accumulator
	—	Place the result of the operation in the following location
	Skip	Specifies the skip test
	High-order word	The most significant 16 bits (bits 16-31)
	Low-order word	The least significant 16 bits (bits 0-15)
	High-order byte	The most significant 8 bits (bits 8-15)
	Low-order byte	The least significant 8 bits (bits 0-7)

Indirect Bit - @

The indirect bit is the first bit (bit 0) of an address. When the bit is SET(1), the address points to a memory location that contains a second address. The second address location might contain a third address, or it might contain data. The address chain can be up to 16 addresses long, as long as the indirect bit of each successive address is set.

When the indirect bit is CLEAR(0), the address points to a memory location containing data.

Control Field - f

The control field, *f*, manipulates bits 8 and 9 of the I/O instructions **DIA, DIB, DIC, DOA, DOB,** and **DOC**. The state of these bits determines the state of the Busy and Done flags of an I/O device.

Carry bit - c

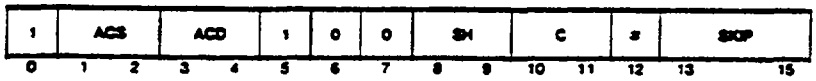
The arithmetic and logic unit (ALU) begins its manipulation of the data by determining an initial value for the carry bit. This new value is based upon three things: the old value of the carry bit, bits 10-11 of the ALC instruction, and which ALC instruction is being executed. The ALU first determines the effect of bits 10-11 on the old value of carry. Table A.3 shows each of the mnemonics that can be appended to the instruction mnemonic, the value of bits 10-11 for each choice, and the action each bit combination causes.

Table A-2. Carry Mnemonics

Symbol	Carry(c) Bit Value	Operation
Omitted	00	Leave carry unchanged.
Z	01	Initialize carry to 0.
O	10	Initialize carry to 1.
C	11	Complement carry.

Add Complement (ADC)

ADC *[c][sh][#]acs.acd[,skip]*



Summary

$\overline{ACS} + ACD \rightarrow ACD$

Detailed Description

Use: To add the logical complement of an unsigned integer to another unsigned integer.

Effect: Sets carry bit to the specified value.

Adds the logical complement of the unsigned, 16-bit integer in ACS to the unsigned, 16-bit integer in ACD. If the result is greater than 2^{16}-1 ($65{,}535_{10}$), then the value of the carry is complemented.

Places the 17-bit value (the carry bit and the result of the add) in the shifter. Performs the specified shift operation. If the no-load bit is zero, loads the 17-bit value into the carry bit and ACD.

Tests the skip condition. If it is true, skips the next sequential word.

Notes

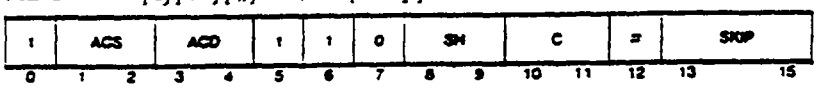
- If the number in ACS is less than the number in ACD, the instruction complements the value of carry before shifting.

Example

Operation	Syntax	Before	After
Add the complement of 010101_8 to 345_8.	ADC 1,0	AC0 = 000345_8 AC1 = 010101_8 Carry = 0	AC0 = 170243_8 AC1 = 010101_8 Carry = 0

Add (ADD)

ADD *[c][sh][#]acs,acd[,skip]*



Summary

ACS + ACD — shifter

Detailed Description

Use: To add unsigned integers.

Effect: Sets carry to the specified value.

Adds the unsigned, 16-bit number in ACS to the unsigned, 16-bit number in ACD. If the result is greater than 2^{16}-1 ($65,535_{10}$), the value of the carry is complemented.

Places the 17-bit value (carry and the result of the add) into the shifter. Performs the specified shift operation. If the no-load bit is zero, places the 17-bit value into the carry bit and ACD.

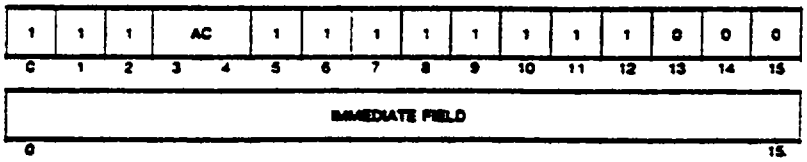
Tests the skip condition. If the condition is true, skips the next sequential word.

Example

Operation	Syntax	Before	After
Add 345_8 and 010101_8.	**ADD** 1,0	AC0 = 000345_8 AC1 = 010101_8 Carry = 0	AC0 = 170243_8 AC1 = 010101_8 Carry = 0

Extended Add Immediate (ADDI)

ADDI *i,ac*



Summary

(Immediate field) + *ac* → *ac*

Detailed Description

Use: To add a signed integer in the range -32678_{10} to $+32767_{10}$ to the contents of an accumulator.

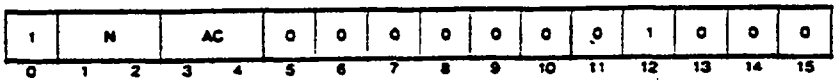
Effect: Adds the signed two's complement number in the immediate field to the signed two's complement number in the specified accumulator. Places the result in the accumulator. Carry remains unchanged.

Example

Operation	Syntax	Before	After
Add 303_8 and 345_8.	**ADDI** 303,1	AC0 = 000345_8	AC0 = 000650_8

Add Immediate (ADI)

ADI *n,ac*



Summary

[*n*+1] + AC — AC

Detailed Description

Use: To add an unsigned integer in the range of 1 to 4 to the contents of an accumulator.

Effect: Adds the unsigned, 16-bit number in the specified accumulator to the contents of the immediate field n plus 1. Carry remains unchanged.

Notes

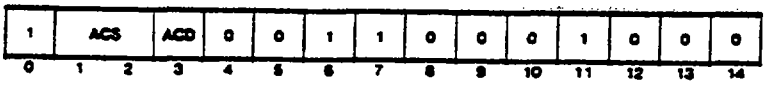
- DGC assemblers compute N before loading the immediate field. You code *n*, which is N + 1. Code the exact value you want to add.

Example

Operation	Syntax	Before	After
Add 4 to 177775_8.	ADI 4,2	AC2 = 177775_8	AC2 = 000001_8

And with Complemented Source (ANC)

ANC *acs,acd*



Summary

(The complement of ACS) + ACD → ACD

Detailed Description

Use: To form the logical AND of the complement of ACS's contents with ACD's contents.

Effect: Sets carry to the specified value. Sets a bit in the result to one if the corresponding bit position in ACS is zero and the corresponding bit position in ACD is one. Otherwise the bits are set to zero.

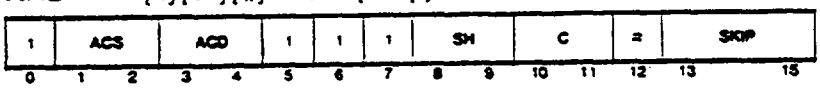
It places the result in ACD and leaves ACS unchanged.

Example

Operation	Syntax	Before	After
AND the complement of AC0 with AC1.	ANC 0,1	AC0 = 177775_8	AC0 = 177775_8

And (AND)

AND *[c][sh][#]acs,acd[,skip]*



Summary

ACS + ACD — ACD

Detailed Description

Use: To form the logical AND of the contents of two accumulators.

Effect: Sets the value of carry to the specified value.

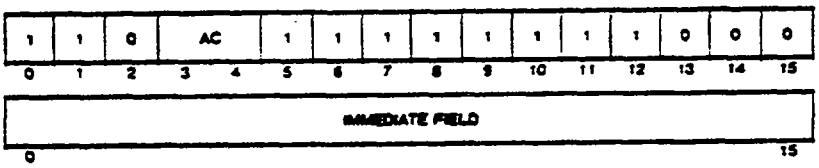
Places carry and the logical AND of ACS and ACD into the shifter. Each bit placed in the shifter is one only if the corresponding bit in both ACS and ACD is one. Otherwise, the bit is zero.

Performs the specified shift operation and places the result in carry and ACD if the no-load bit is zero.

If the skip condition is true, the next sequential word is skipped.

And Immediate (ANDI)

ANDI *i,ac*



Summary

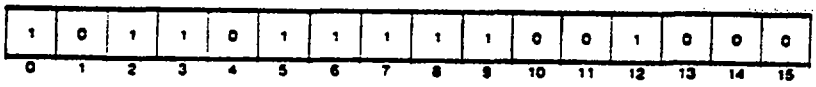
(Immediate field) + AC — AC

Detailed Description

Effect: Forms the logical AND of the contents of the immediate field and the contents of the specified accumulator.

Block Move (BLM)

BLM



Summary

(Memory) ←(Memory)

Detailed Description

Use: To move memory words from one location to another.

Effect: Memory words are moved sequentially. They are treated as unsigned 16-bit integers. The address of the source location is contained in bits 1-15 of AC2.

The address of the destination is contained in bits 1-15 of AC3. If bit zero of either AC2 or AC3 is one, then the address in bits 1-15 is assumed to be indirect. Before the data movement occurs, the indirection chain of finding addresses is followed, and the resulting effective address is placed in the accumulator.

The number of words moved is equal to the unsigned word in AC1. This number must be greater than 0 and less than or equal to 100000_8. If the number contained in AC1 is outside these bounds, no data is moved and the contents of the accumulators are unchanged.

AC	Contents
1	Number of words to be moved
2	Source address
3	Destination address

For each word moved, the count in AC1 is decremented by one and the source and destination addresses in AC2 and AC3 are incremented by one. Upon completion of the instruction, AC1 contains all zeros, and AC2 and AC3 point to the word following the last word in their respective fields.

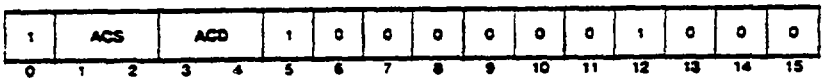
Words are moved in consecutive, ascending order according to their addresses. The next address after 77777_8 is 0 for both fields. The fields may overlap in any way.

Notes

- Since this instruction can take a long time (compared to I/O requests), it can be interrupted. If an interrupt occurs, the current location of memory will be saved, so it can be returned to upon completion of the interrupt service routine.
- When an interrupt occurs, the program counter (PC) is decremented by one and then stored in location zero. Since the addresses and the word count are updated after every word storage, any interrupt service routine that returns the program to the address in location zero will restart the transferring of words in the correct location.
- When updating the source and destination addresses, the BLM instruction forces bit 0 of the result to zero. This ensures that upon return from an interrupt, the instruction will not try to resolve an indirect address in either AC2 or AC3.

Set Bit to One (BTO)

BTO *acs,acd*



Summary

1 — (a bit in memory)

Detailed Description

Use: To form a bit pointer from the contents of two accumulators.

Effect: Forms a bit pointer from ACS and ACD. ACS contains the high-order word of the bit pointer and ACD contains the low-order word. If you specify ACS and ACD as the same accumulator, the contents of that accumulator become the low-order word of the bit pointer. The high-order word is zero.

Sets the addressed bit in memory to one. Leaves ACS and ACD unchanged.

Notes

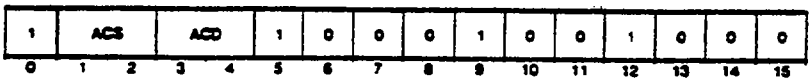
- Bit zero of the bit pointer must be zero. The bit pointer contained in ACS and ACD must not make an indirect memory reference.

Example

Operation	Syntax	Before	After
Set bit 7 at 23456_8 to 1.	**BTO** 0,1	AC0 = 023450_8 AC1 = 000147_8 23456_8 = 010103_8	AC0 = 023450_8 AC1 = 000147_8 23456_8 = 010503_8

Set Bit to Zero (BTZ)

BTZ *acs,acd*



Summary

0 — (bit in memory)

Detailed Description

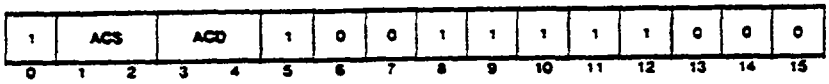
Use: To set the addressed bit to zero.

Effect: ACS and ACD are combined to form a 32-bit pointer. ACS is the high-order word and ACD is the low-order word. If ACS and ACD are specified as the same accumulator, then the 32-bit pointer is treated as having its high-order word equal to zero and its low-order word equal to the accumulator that contains either ACS or ACD.

The instruction then sets the addressed bit in memory to zero, leaving the contents of ACS and ACD unchanged.

Compare to Limits (CLM)

CLM *acs,acd*



Summary

Skips if between two values: L and H

Detailed Description

Use: To compare a signed integer with two other integers and skip if the first is between the other two.

Effect: Compares the signed two's complement integer contained in ACS to L and H. (L and H are two signed two's complement limit values.) If the number in ACS is greater than or equal to L and less than or equal to H, the next sequential word is skipped.

If ACS and ACD are specified as different accumulators, the address of the limit value L is contained in bits 1-15 of ACD. The limit value H is contained in the word following L. Bit 0 of ACD is ignored.

If ACS and ACD are specified as the same accumulator, then the number to be compared is contained in that AC and the limit values L and H are contained in the two words following the instruction. L is the first word and H is the second word. The next sequential word is the third word following the instruction.

Example

Operation: Compare 000331_8 to L stored at 001234_8 and H stored at 001235_8.

Instructions: **CLM** 1,0

Skipped instruction
Executed instruction

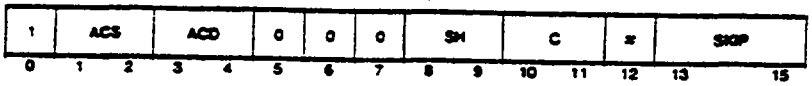
Accumulators: AC0 = 001234_8
AC1 = 000331_8

Locations: 001234_8 = 000017_8
001235_8 = 001112_8

Result: Since the number in AC1 is between the limit values, the processor skips the next sequential word.

Complement (COM)

COM *[c][sh][#]acs,acd[,skip]*



Summary

$\overline{ACS}$ — ACD

Detailed Description

Use: To form the logical complement of the contents of an accumulator.

Effect: Sets the carry to the specified value.

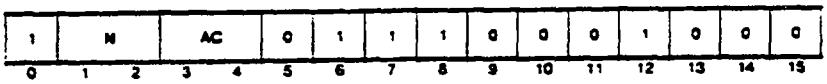
Forms the logical complement of the number in ACS and places the 17-bit value (carry and function result) in the shifter.

Performs the specified shift operation and places the result in carry and ADC if the no-load bit is zero.

If the skip condition is true, the next sequential word is skipped.

Double Hex Shift Left (DHXL)

DHXL *n,ac*



Summary

(AC, AC+1) — shifted left n digits

Detailed Description

Use: To shift a double word left.

Effect: Shifts the word comprised of AC (high-order word) and AC + 1 (low-order word) left N + 1 hex digits.

Bits shifted out are lost, and vacated bit positions become zeros. If you specify AC as AC3, AC + 1 is AC0. Carry remains unchanged.

Notes

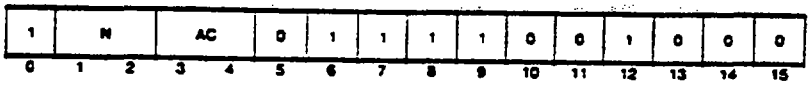
- DGC assemblers compute N. You code *n*, which is N + 1. The assembler takes the coded value of *n* and subtracts one from it before placing it in the immediate field. Therefore, you should code the exact number of hex digits that you want to shift.
- If N is equal to 3, the contents of AC are placed in A+1 and AC is filled with zeros.

Example

Operation	Syntax	Before	After
Shift the 32-bit number contained in AC0 and AC1 left one hex digit.	**DHXL** 1,0	AC0 = 001160_8 AC1 = 050010_8	AC0 = 023405_8 AC1 = 000200_8

Double Hex Shift Right (DHXR)

DHXR *n,ac*



Summary

(AC, AC+1) — shifted right *n* digits

Detailed Description

Use: To shift a double word right.

Effect: Shifts the 32 bits contained in AC (high-order word) and AC + 1 (low-order word) right N + 1 hex digits.

Bits shifted out are lost, and the vacated bit positions are filled with zeros. If you specify AC as AC3, AC + 1 is AC0. Carry remains unchanged.

Notes

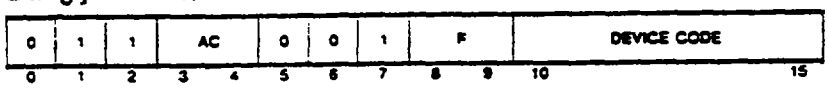
- DGC assemblers compute N. You code *n*, which is N + 1. Code the exact number of hex digits you want shifted. If *n* is equal to four, all of AC shifts into AC + 1. AC + 1 fills with zeros.

Example

Operation	Syntax	Before	After
Divide 23450000103_8 by 256_{10}. (Same as shifting right two hex digits.)	**DHXR** 2,0	AC0 = 001160_8 AC1 = 050010_8	AC0 = 023405_8 AC1 = 000200_8

Data In A (DIA)

DIA[*f*] *ac.device*



Summary

Moves the A buffer of an I/O device — AC

Detailed Description

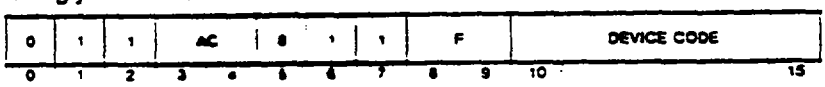
Use: To transfer data from the A buffer of an I/O device to an accumulator.

Effect: The contents of the A input buffer in the specified device are placed in the specified AC. The operation sets the Busy and Done flags according to the function defined by *f*.

The format of the AC after the transfer is device dependent. If the specified device does not exist, the AC will contain 177777_8 after the transfer.

Data In B (DIB)

DIB[*f*] *ac.device*



Summary

Moves the B buffer of an I/O device — AC

Detailed Description

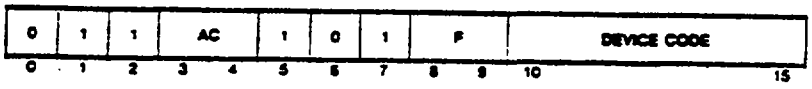
Use: To transfer data from the B buffer of an I/O device to an accumulator.

Effect: The contents of the B input buffer in the specified device are placed in the AC. Sets the Busy and Done flags according to the function defined by *f*.

The format of the AC after the transfer is device dependent. If the specified device does not exist, the AC will contain 177777_8 after the transfer.

Data In C (DIC)

DIC[*f*] *ac,device*



Summary

Moves the C buffer of an I/O device — AC

Detailed Description

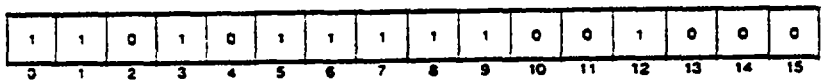
Use: To transfer data from the C buffer of an I/O device to an accumulator.

Effect: The contents of the C input buffer in the specified device are placed in the AC. The operation sets the Busy and Done flags according to the function defined by *f*.

The final format of the AC is device dependent. If the specified device does not exist, the AC will contain 177777_8 after the transfer.

Unsigned Divide (DIV)

DIV



Summary

(AC0, AC1) / AC2

Detailed Description

Use: To divide the unsigned 32-bit integer in AC0 (high-order word) and AC1 (low-order word) by the unsigned 16-bit number in AC2.

Effect: The quotient and remainder are both unsigned 16-bit numbers. The quotient occupies AC1 and the remainder is in AC0.

It sets carry to zero and leaves AC2 unchanged.

Notes

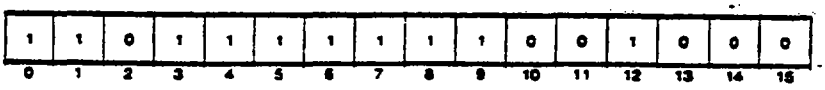
- Before the divide operation, the number in AC0 is compared (unsigned) to the number in AC2. If the contents of AC0 are greater than or equal to the contents of AC2, an overflow condition is indicated. The carry bit is set to 1 and the operation is terminated. All operands remain unchanged.

Examples

Operation	Syntax	Before	After
Divide 6 by 2.	DIV	AC0 = 000000_8 AC1 = 000006_8 AC2 = 000002_8 Carry = 0	AC0 = 000000_8 AC1 = 000003_8 AC2 = 000002_8 Carry = 0
Divide 7777600001_8 by 077777_8.	DIV	AC0 = 037777_8 AC1 = 000001_8 AC2 = 077777_8 Carry = 0	AC0 = 000000_8 AC1 = 077777_8 AC2 = 0 Carry = 0
Divide $3777740000 1_8$ by 177777_8. Overflow results.	DIV	AC0 = 177776_8 AC1 = 000001_8 AC2 = 077777_8 Carry = 0	AC0 = 177776_8 AC1 = 000001_8 AC2 = 077777_8 Carry = 1
Divide 7 by 2. Get a remainder of 1.	DIV	AC0 = 000000_8 AC1 = 000007_8 AC2 = 000002_8 Carry = 0	AC0 = 000001_8 AC1 = 000003_8 AC2 = 000002_8 Carry = 0

Signed Divide (DIVS)

DIVS



Summary

(AC0,AC1) / AC2

Detailed Description

Use: To divide the 32-bit two's complement number comprised of AC0 and AC1 by the 16-bit two's complement number in AC2. AC0 is the high-order word and AC1 is the low-order word.

Effect: The quotient and remainder are both 16-bit, two's complement numbers. The quotient occupies AC1, and the remainder occupies AC0. The rules of algebra determine the quotient's sign. The remainder's sign is always the same as the dividend's sign, except that a zero quotient or a zero remainder is always positive.

Sets carry to zero. Leaves AC2 unchanged.

Notes

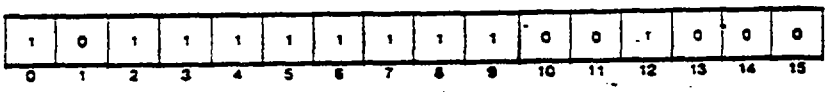
- If the quotient is too large to fit into AC1, an overflow results. The carry bit is set to 1 and the instruction is terminated. The contents of AC0 and AC1 are unpredictable.

Examples

Operation	Syntax	Before	After
Divide 6 by 2.	DIVS	AC0 = 000000_8 AC1 = 000006_8 AC2 = 000002_8	AC0 = 000000_8 AC1 = 000003_8 AC2 = 000002_8
Divide 7777600001_8 by 077777_8.	DIVS	AC0 = 037777_8 AC1 = 000001_8 AC2 = 077777_8	AC0 = 000000_8 AC1 = 077777_8 AC2 = 077777_8
Divide 1 by -1.	DIVS	AC0 = 000000_8 AC1 = 000001_8 AC2 = 177777_8	AC0 = 000000_8 AC1 = 177777_8 AC2 = 177777_8
Divide -7 by -2. Get a remainder of -1.	DIVS	AC0 = 177777_8 AC1 = 177777_8 AC2 = 177777_8	AC0 = 177777_8 AC1 = 000003_8 AC2 = 177777_8

Sign Extend and Divide (DIVX)

DIVX



Summary

Gives both accumulators' contents the same sign, then executes a DIVS with the results.

Detailed Description

Use: To extend the sign of one accumulator into a second accumulator and perform a **DIVS** operation on the result.

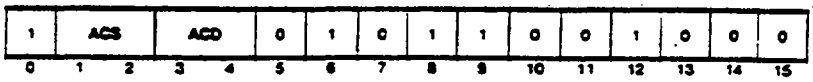
Effect: Extends the sign of the number in AC1 into AC0 by placing a copy of bit 0 of AC1 in each bit of AC0. After the sign extension, a **DIVS** is performed.

Notes

- If the quotient is too large to fit into AC1, an overflow results. The processor sets carry to 1 and terminates the instruction operation. The contents of AC0 and AC1 are unpredictable.

Double Logical Shift (DLSH)

DLSH *acs,acd*



Summary

Shifts (ACD,ACD+1) left or right, depending on bits 8 to 15 in ACS.

Detailed Description

Use: To logically shift a 32-bit word left or right.

Effect: The 32-bit word formed by ACD (high-order word) and ACD+1 (low-order word) is shifted left or right, depending on the number in bits 8-15 in ACS. AC3+1 is AC0.

The signed two's complement byte in bits 8-15 of ACS determines the direction of the shift and the number of bits to be shifted. If the number is positive, the shift is to the left. If the number is negative, the shift is to the right. If the number is zero, no shifting is performed. Bits 0-7 of ACS are unchanged.

The number of bits shifted equals the high-order word of ACS. Bits shifted out are lost and the vacated bit positions are filled with zeros. The carry bit and the contents of ACS remain unchanged, unless ACS is ACD+1.

Notes

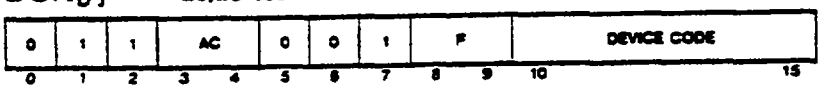
- If the high-order word of ACS is greater than 31_{10}, all bits of ACD are cleared. The carry bit and the contents of ACS remain unchanged. If ACD is specified as AC3, then ACD+1 is AC0.

Examples

Operation	Syntax	Before	After
Shift left one bit.	DLSH 0,1	AC0 = 000001_8 AC1 = 012345_8 AC2 = 054321_8	AC0 = 000001_8 AC1 = 024712_8 AC2 = 130642_8
Shift right one bit.	DLSH 0,1	AC0 = 000377_8 AC1 = 024712_8 AC2 = 000000_8	AC0 = 000377_8 AC1 = 012345_8 AC2 = 000000_8

Data Out A (DOA)

DOA[*f*] *ac,device*



Summary

Transfer from an accumulator — to the A buffer of an I/O device

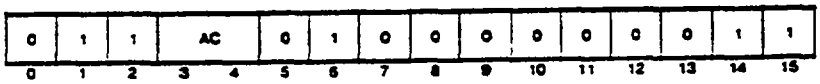
Detailed Description

Use: To transfer data from an accumulator to the A buffer of an I/O device.

Effect: Places the contents of the specified AC in the A output buffer of the specified device. Sets the Busy and Done flags according to the function specified by *f*. The contents of the specified accumulator remain unchanged.

Load MAP Status (DOA MAP)

DOA *ac,* **MAP**



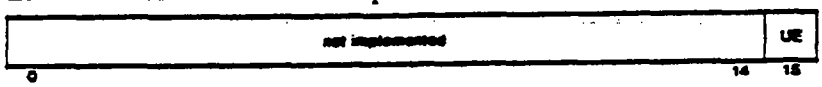
Summary

AC MAP status register

Detailed Description

Use: The contents of the specified AC are placed in the MAP status register. The contents of the AC remain unchanged.

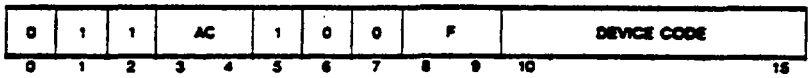
Effect: The format of the specified AC is:



When bit 15 is set, mapping of the CPU addresses will start with the first memory reference after the next indirect reference or a return type instruction (**POPB, POPJ, RTN, RSTR**).

Data Out B (DOB)

DOB[*f*] *ac,device*



Summary

Transfer from an accumulator — to the B buffer of an I/O device

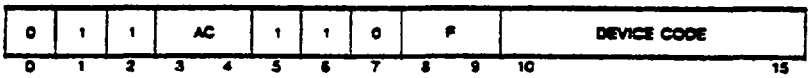
Detailed Description

Use: To transfer data from an accumulator to the B buffer of an I/O device.

Effect: Places the contents of the specified AC in the B output buffer of the specified device. Sets the Busy and Done flags according to the function specified by *f*. The contents of the specified accumulator remain unchanged.

Data Out C (DOC)

DOC[*f*] *ac,device*



Summary

Transfer from an accumulator — to the C buffer of an I/O device

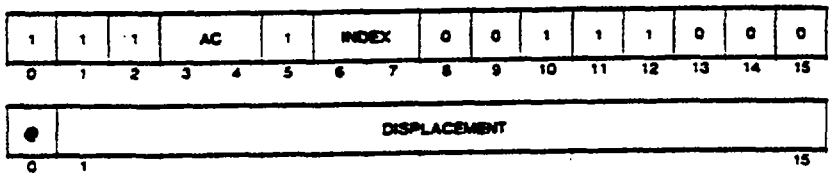
Detailed Description

Use: To transfer data from an accumulator to the C buffer of an I/O device.

Effect: Places the contents of the specified AC in the C output buffer of the specified device. Sets the Busy and Done flags according to the function specified by *f*. The contents of the specified accumulator remain unchanged.

Dispatch (DSPA)

DSPA *ac*



Summary

The dispatch table is processed.

Detailed Description

Use: To conditionally transfer control to a location whose address is selected from a table in memory (see Figure A-1).

Effect: Computes the effective address E. This is the address of H, the high limit, and L, the low limit. Each is a two's complement number. H is in location E-1, and L is in location E-2. The last entry in the dispatch table is in location E + H - L.

Compares the two's complement number in the specified accumulator to the limit values. If the number is less than L or greater than H, operation continues with the word immediately following the **DSPA** instruction. If the number is greater than L and less than or equal to H, the processor fetches the word at location E - L + number.

If the fetched word is 177777_8, operation continues with the word immediately following the **DSPA** instruction. Otherwise, the fetched word is the intermediate address in an effective address calculation.

After resolving the indirection, if any, the instruction places the effective address in the program counter. Operation continues with the word addressed by the updated program counter.

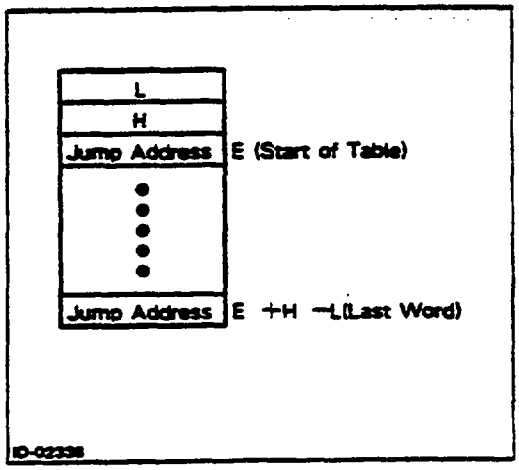
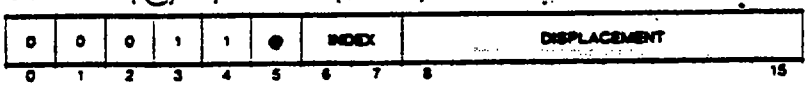


Figure A-1. DSPA Operation

Decrement And Skip if Zero (DSZ)

DSZ *[@]displacement[,index]*



Summary

Decrement location addressed by *E*; skip the next word if result = 0.

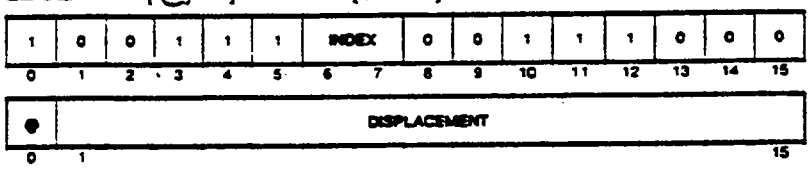
Detailed Description

Use: To decrement an addressed word and skip if the decremented value is zero.

Effect: Computes the effective address. Decrements the addressed word by one and writes the result back into that location. If the updated value of the location is zero, the instruction skips the next sequential word. The CPU controls the memory bus until the instruction is completed.

Extended Decrement and Skip if Zero (EDSZ)

EDSZ *[@]displacement[,index]*



Summary

Decrement location addressed by *E*; skip the next word if result = 0.

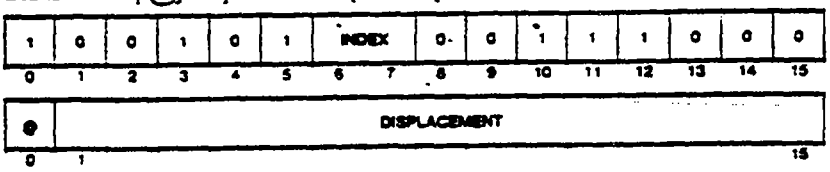
Detailed Description

Use: To decrement the addressed word and skip if the decremented value is zero.

Effect: Computes the effective address. Decrements the addressed word by one and writes the result back into that location. If the updated value of the location is zero, the instruction skips the next sequential word. The CPU controls the memory bus until the instruction is completed.

Extended Increment and Skip if Zero (EISZ)

EISZ *[@]displacement[,index]*



Summary

Increment the word addressed by *E*; if the result = 0, skip the next word.

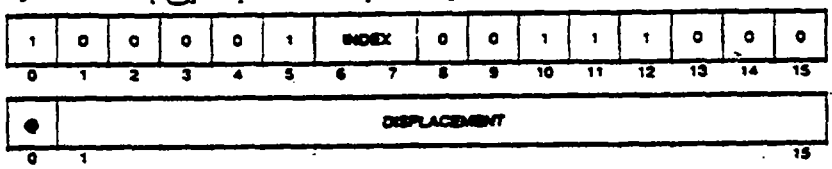
Detailed Description

Use: To increment the addressed word. Skips if the incremented value is zero.

Effect: Computes the effective address. Increments the contents of the location specified by the effective address, and writes the new value back into memory at the same address. If the updated value of the location is zero, the instruction skips the next sequential word. The CPU controls the memory bus until the instruction is completed.

Extended Jump (EJMP)

EJMP *[@]displacement[,index]*



Summary

(New location) — PC

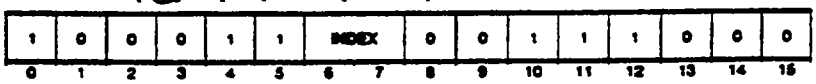
Detailed Description

Use: Places an effective address in the program counter (PC).

Effect: The effective address, *E*, is computed and placed in the program counter. The program continues at the new location indicated by the program counter.

• Extended Jump to Subroutine (EJSR)

EJSR *[@]displacement[,index]*



Summary

(Effective address) → PC

Detailed Description

Use: To increment and store the value of the program counter in AC3, then place a new address in the program counter.

Effect: Computes the effective address. Places the address of the next sequential word (the word following the JSR instruction) in AC3. Places the effective address in the program counter. Operation continues with the word addressed by the updated program counter.

Notes

- The instruction computes the effective address before it places the incremented program counter in AC3.

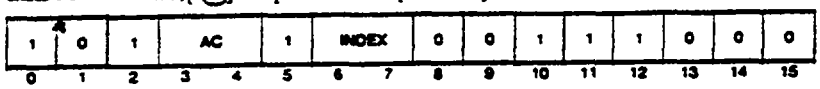
Example

Instruction:
```
EJSR SUBR
Next instruction
.
Next instruction
.
.
SUBR:   First SUBR (subroutine) instruction
```

After execution of the **EJSR** instruction, the program continues operation with the first **SUBR** instruction. AC3 contains the address of the next instruction after the **EJSR** operation.

Extended Load Accumulator (ELDA)

ELDA *ac,[@]displacement[,index]*



Summary

Memory word — AC

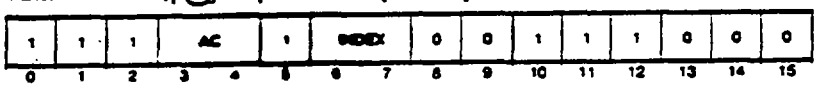
Detailed Description

Use: To move a copy of the contents of a memory word into the specified accumulator.

Effect: Calculates the effective address. Places the contents of the addressed location in the specified accumulator. The addressed location remains unchanged.

Load Effective Address (ELEF)

ELEF *ac,[@]displacement[,index]*



Summary

(Effective address E) — (bits 1-15) of AC

Detailed Description

Use: To place an effective address in an accumulator.

Effect: Computes the effective address and places it in bits 1-15 of the specified accumulator. Sets bit 0 of the accumulator to zero. The previous contents of the accumulator are overwritten.

Example

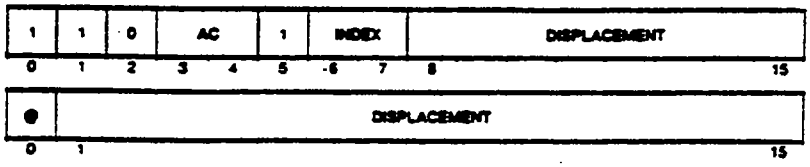
```
ELEF 0,TABLE      ;The logical address of TABLE
                  ;is placed in AC0.

ELEF 1,-55,3      ;Subtracts 000055₈ from the unsigned integer
                  ;in AC3 and places the result in AC1.

ELEF 0,+0         ;Places the logical address of this ELEF
                  ;instruction in AC0.
```

Extended Store Accumulator (ESTA)

ESTA *ac.[@]displacement[.index]*



Summary

(AC) — memory location

Detailed Description

Use: To store the contents of an accumulator in a memory location.

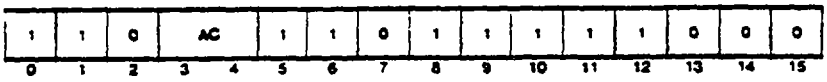
Effect: Places the contents of the specified accumulator in the word addressed by the effective address. Overwrites the previous contents of the addressed location. The contents of the specified accumulator remain unchanged.

Example

Operation	Syntax	Before	After
Store the contents of location AC1 into memory location 001123_8.	ESTA 1,1123	AC1 = 010101_8 001123_8 = xxxxxx	AC1 = 010101_8 001123_8 = 010101_8

Halve (HLV)

HLV *ac*



Summary

AC / 2 — AC

Detailed Description

Use: To half the 16-bit two's complement number in the specified accumulator.

Effect: Halves the 16-bit two's complement number in the specified accumulator. Truncates the result and stores it in the specified accumulator.

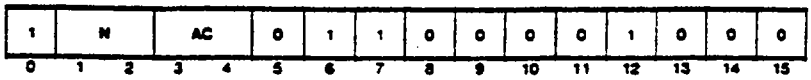
If the number in the accumulator is positive, the instruction performs the division by shifting the number right one bit. If the number in the accumulator is negative, the instruction performs the division by negating the number, shifting it right one bit and then negating it again.

Example

Operation	Syntax	Before	After
Divide 4523_8 by 2.	HLV 0	AC0 = 004523_8	AC0 = 002251_8
Divide -2 by 2.	HLV 1	AC1 = 177776_8	AC1 = 177777_8

Hex Shift Left (HXL)

HXL *n,ac*



Summary

Shift the AC left N+1 digits.

Detailed Description

Use: Shift the contents of an accumulator left.

Effect: Shifts the contents of the specified accumulator left N+1 hex digits. Bits shifted out are lost. Vacated positions become zeros.

Notes

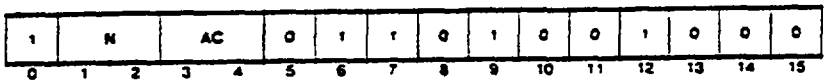
- DGC assemblers compute N. You code *n*, which is N+1. Code the exact number of hex digits you want shifted. If *n* = 4, AC fills with zeros.

Example

Operation	Syntax	Before	After
Shift the 16-bit number in AC1 left 2 hex digits.	HXL 2,1	AC1 = 004523_8	AC0 = 051400_8

Hex Shift Right (HXR)

HXR *n,ac*



Summary

Shift the AC right N+1 digits.

Detailed Description

Use: Shift the contents of an accumulator right.

Effect: Shifts the contents of the specified accumulator right N+1 hex digits. Bits shifted out are lost. Vacated positions become zeros.

Notes

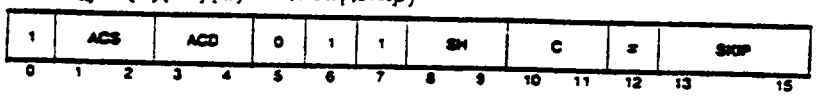
- DGC assemblers compute N. You code in *n*, which is N+1. Code the exact number of hex digits you want shifted. If *n* = 4, AC fills with zeros.

Example

Operation	Syntax	Before	After
Shift the 16-bit number in AC1 right 2 hex digits.	**HXL** 2,1	AC1 = 004523_8	AC0 = 000011_8

Increment (INC)

INC *[c][sh][#]acs,acd[,skip]*



Summary

ACS + 1 — ACD

Detailed Description

Use: To increment the contents of an accumulator.

Effect: Sets carry to the specified value.

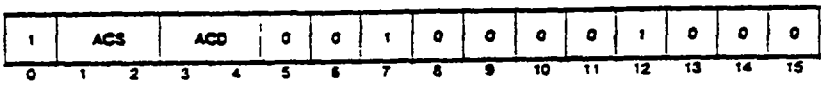
Increments the unsigned, 16-bit number in ACS by one. If the result is greater than $2^{16}-1$ ($65{,}535_{10}$, the value of the carry is complemented.

Places the 17-bit value (carry and the result of the increment) into the shifter. Performs the specified shift operation. If the no-load bit is zero, loads the 17-bit value into the carry bit and ACD.

Tests the skip condition. If the condition is true, skips the next sequential word.

Inclusive OR (IOR)

IOR *ac,acd*



Summary

ACS + ACD → ACD

Detailed Description

Use: To form the logical inclusive OR of the contents of two accumulators.

Effect: Forms the logical inclusive OR of ACS and ACD. Places the result in ACD.

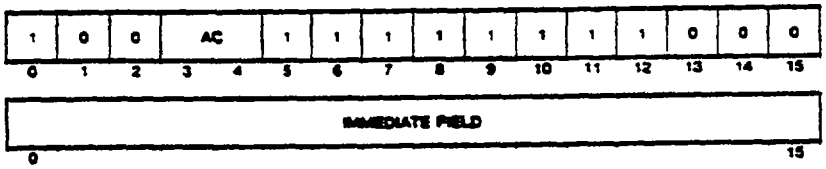
This instruction sets a bit position in ACD to 1, if the corresponding bit positions in ACD and ACS are not both zero.

Example

Operation	Syntax	Before	After
Inclusive OR the contents of AC0 with the contents of AC1.	**IOR** 0,1	AC0 = 002245_8 AC1 = 010133_8	AC0 = 002245_8 AC1 = 012377_8

Inclusive OR Immediate (IORI)

IORI *i,ac*



Summary

(Immediate field) + AC → AC

Detailed Description

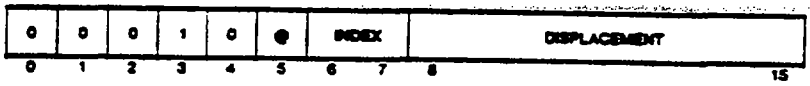
Use: To form the logical inclusive OR of the contents of an immediate field and an accumulator.

Effect: Forms the logical inclusive OR of the contents of the immediate field and of the specified AC, and places the result in the specified AC.

This instruction sets a bit in the specified accumulator to 1, if the corresponding bit positions in the accumulator and the immediate field are not both zero.

Increment and Skip if Zero (ISZ)

ISZ *[@]displacement[,index]*



Summary

Increment the word addressed by *E*; if the result = 0, skip the next word.

Detailed Description

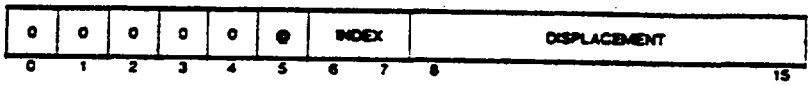
Use: To increment the addressed word and skip if the incremented value is zero.

Effect: Computes the effective address. Increments the addressed word by one. Stores the result in the same location. If the incremented value is zero, the instruction skips the next sequential word.

The CPU controls the memory bus until the instruction is completed.

Jump (JMP)

JMP *[@]displacement[,index]*



Summary

New location — PC

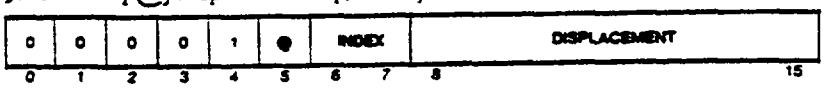
Detailed Description

Use: Places an effective address in the program counter.

Effect: The effective address, *E*, is computed and placed in the program counter. The program continues at the new location indicated by the program counter.

Jump to Subroutine (JSR)

JSR *[@]displacement[,index]*



Summary

Effective address — PC

Detailed Description

Use: To perform a jump to a subroutine.

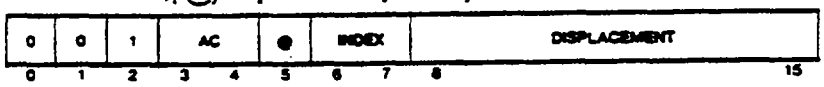
Effect: Computes the effective address, then places the address of the next sequential word in AC3. Places the effective address in the program counter. Operation continues with the word addressed by the program counter.

Notes

- The instruction computes the effective address before it places the incremented program counter in AC3.

Load Accumulator (LDA)

LDA *ac,[@]displacement[,index]*



Summary

(Address *E*) — AC

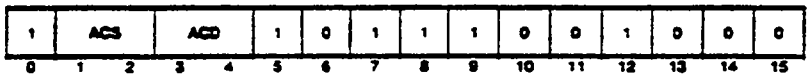
Detailed Description

Use: To copy a word from memory to an accumulator.

Effect: Computes the effective address *E*. Places the word addressed by the effective address into the specified accumulator. The contents of the addressed location remain unchanged.

Load Byte (LDB)

LDB *acs,acd*



Summary

(Byte in memory) — high byte of ACD

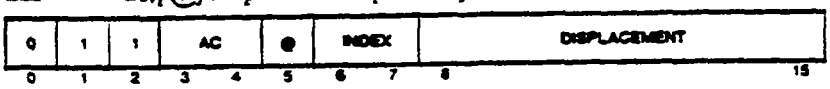
Detailed Description

Use: Places a byte of information into an accumulator.

Effect: Uses the byte pointer contained in ACS to load a byte from memory into bits 8 to 15 of ACD. Sets bits 0 to 7 of ACD to zero. The contents of ACS remain unchanged, unless ACS and ACD are the same accumulator.

Load Effective Address (LEF)

LEF *ac,[@]displacement[,index]*



Summary

Computes an effective address — AC

Detailed Description

Use: To compute an effective address and place it into an accumulator.

Effect: Places the calculated effective address into bits 1 to 15 of the specified AC. Bit 0 of the AC is set to zero.

With the **LEF** mode bit set to one, all I/O and **LEF** instructions will be interpreted as **LEF** instructions.

With the **LEF** mode bit set to zero, all I/O and **LEF** instructions will be interpreted as I/O instructions.

Be sure that I/O protection is enabled, or that the **LEF** mode bit is set to one before using the **LEF** instruction. If you issue an **LEF** instruction in the I/O mode with protection disabled, the instruction will be interpreted and executed as an I/O instruction, with possible undesirable results.

Notes

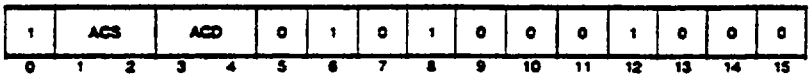
- The LEF instruction can be used only in a mapped system.

Example

```
LEF     0,TABLE     ;The logical address of TABLE is placed in AC0.
LEF     1,-55,3     ;Subtracts 000055(8) from the unsigned 15-bit
                    integer in AC3 and the result is placed in AC1.
LEF     0,.+0       ;Places the address of this LEF instruction in AC0.
```

Logical Shift (LSH)

LSH *acs,acd*



Summary

Shift ACD either — or —

Detailed Description

Use: To shift a number left or right.

Effect: Shift the 16-bit number in ACD either left or right.

The sign of the 8-bit two's complement number in bits 8-15 of ACS determines the direction of the shift. If this number is positive, the shift is to the left. If it is negative, the shift is to the right. Bits 0-7 of ACS remain unchanged.

The magnitude of the 8-bit two's complement number in bits 8-15 of ACS determines the number of bits to be shifted. Bits shifted out are lost. Vacated bit positions become zero.

The carry bit and ACS remain unchanged.

Notes

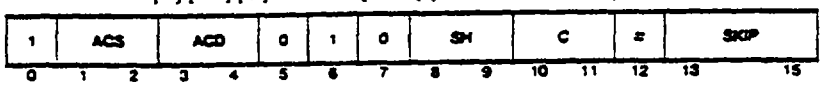
- If the magnitude of the number in bits 8-15 of ACS is greater than 15, all bits of ACD become zero.

Example

Operation	Syntax	Before	After
Shift left one bit	LSH 0,1	AC0 = 000001_8 AC1 = 012345_8	AC0 = 000001_8 AC1 = 024712_8
Shift right one bit	LSH 0,1_8	AC0 = 0003777_8 AC1 = 024712_8	AC0 = 000377_8 AC1 = 012345_8

Move (MOV)

MOV *[c][sh][#]acs,acd[,skip]*



Summary

ACS — ALU — ACD (if no skip is involved)

Detailed Description

Use: To move the contents of an accumulator through the arithmetic logic unit (ALU).

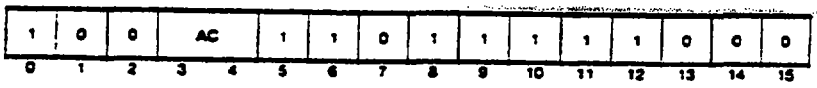
Effect: Sets carry to the specified value.

Places the carry and the contents of ACS in the shifter. Performs the specified shift operation.

Tests the skip condition. If the no-load bit is zero, loads the result of the shift into the carry bit and ACD. If the condition is true, skips the next sequential word.

Modify Stack Pointer (MSP)

MSP *ac*



Summary

AC + SP → SP

Detailed Description

Use: To change the value of the stack pointer and test for potential overflow.

Effect: Adds the 16-bit two's complement number in the specified AC to the stack pointer and places the result in the stack pointer (location 40_8).

Checks for overflow by comparing the number now in location 40_8 to the stack limit. If the new stack pointer is greater than the stack limit, it restores the stack pointer's original value and pushes a return block into the stack. The final stack pointer is the original stack pointer plus five. The return block is in this format:

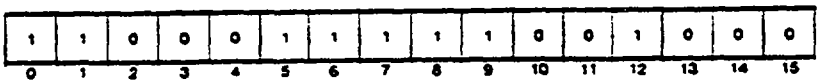
Word Pushed	Contents
1	AC0
2	AC1
3	AC2
4	AC3
5	Bit 0 equals carry; bits 1-15 equal PC

The program counter in the return block contains the address of the MSP instruction.

After pushing the return block, the program counter contains the address of the stack fault routine. Control transfers to the stack fault routine.

Unsigned Multiply (MUL)

MUL



Summary

AC1 * AC2, then add (AC0), — AC0

Detailed Description

Use: To multiply the contents of two accumulators and store the result in a third accumulator.

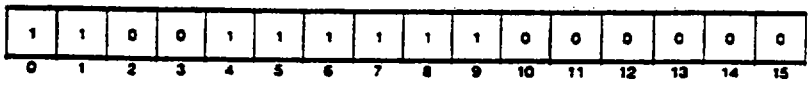
Effect: Multiplies the unsigned 16-bit number in AC1 by the unsigned 16-bit number in AC2 to yield an unsigned 32-bit product. Adds the unsigned 16-bit number in AC0 to the product. The result is an unsigned 32-bit number that occupies AC0 (high-order word), and AC1 (low-order word). AC2 and carry remain unchanged. Because the result is a double-length number, overflow cannot occur.

Examples

Operation	Syntax	Before	After
Multiply 3 by 2.	MUL	AC0 = 000000_8 AC1 = 000003_8 AC2 = 000002_8	AC0 = 000000_8 AC1 = 000006_8 AC2 = 000002_8
Multiply 077777_8 by 177777_8.	MUL	AC0 = 000000_8 AC1 = 077777_8 AC2 = 077777_8	AC0 = 037777_8 AC1 = 000001_8 AC2 = 077777_8
Multiply 177777_8 by 177777_8.	MUL	AC0 = 000000_8 AC1 = 177777_8 AC2 = 177777_8	AC0 = 000000_8 AC1 = 000001_8 AC2 = 077777_8
Multiply 3 by 2 and add a remainder of 1.	MUL	AC0 = 000001_8 AC1 = 000003_8 AC2 = 000002_8	AC0 = 000000_8 AC1 = 000007_8 AC2 = 000002_8

Signed Multiply (MULS)

MULS



Summary

AC1 * AC2, then add AC0, — AC0

Detailed Description

Use: To multiply the contents of two accumulators and place the result in a third accumulator.

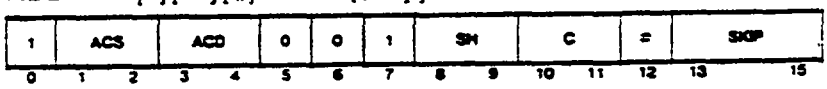
Effect: Multiplies the signed 16-bit two's complement number in AC1 by the 16-bit two's complement number in AC2 to yield a 32-bit two's complement product. Adds the 16-bit two's complement number in AC0 to the product. The result is a 32-bit two's complement number that occupies AC0 (high-order word) and AC1 (low-order word). AC2 and carry remain unchanged. Because the result is a double-length number, overflow cannot occur.

Examples

Operation	Syntax	Before	After
Multiply 3 by 2.	MULS	AC1 = 000003_8 AC2 = 000002_8 AC0 = 000000_8	AC0 = 000000_8 AC1 = 000006_8 AC2 = 000002_8
Multiply 077777_8 by 077777_8.	MULS	AC0 = 000000_8 AC1 = 077777_8 AC2 = 077777_8	AC0 = 037777_8 AC1 = 000001_8 AC2 = 077777_8
Multiply -1 by -1.	MULS	AC0 = 000000_8 AC1 = 177777_8 AC2 = 177777_8	AC0 = 000000_8 AC1 = 000001_8 AC2 = 177777_8
Multiply 3 by -2 and add a remainder of -1.	MULS	AC0 = 177777_8 AC1 = 000003_8 AC2 = 177777_8	AC0 = 177777_8 AC1 = 177777_8 AC2 = 177777_8

Negate (NEG)

NEG *[c][sh][#]acs,acd[,skip]*



Summary

The negative of ACS — ACD

Detailed Description

Use: To form the two's complement of the contents of an accumulator.

Effect: Sets the carry to the specified value.

Takes the two's complement of the number in ACS. Places the 17-bit value (the carry bit and the result of the negation) in the shifter.

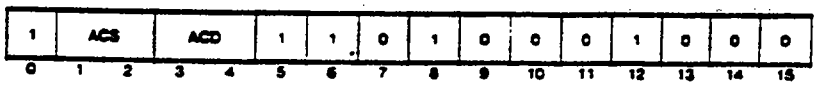
Performs the specified shift operation. If the no-load bit is zero, places the 17-bit value in the carry bit and ACD.

Tests the skip condition. If the condition is true, the next sequential word is skipped.

- If ACS contains zero, the instruction complements the carry.

Pop Multiple Accumulators (POP)

POP *acs,acd*



Summary

SP — AC

Detailed Description

Use: To pop one to four words off the stack and place them in specified accumulators.

Effect: Loads the accumulators in descending order, starting with ACS and ending with ACD. ACD-1 is AC3. If you specify ACS and ACD as the same accumulator, the instruction pops one word off the stack and places it in that accumulator.

Decrements the stack pointer by the number of accumulators loaded. Leaves the frame pointer unchanged.

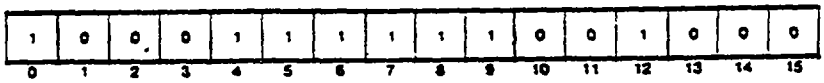
Example

Operation	Syntax	Before	After
To pop 3 words off the stack.	**POP** 3,1	AC0 xxxxxx AC1 xxxxxx AC2 xxxxxx AC3 xxxxxx	Untouched Third word popped Second word popped First word popped

xxxxxx denotes unknown contents, that are overwritten.

Pop Block (POPB)

POPB



Summary

(5 words of SP) — predetermined locations

Detailed Description

Use: To pop five words off the stack and place them in predetermined locations.

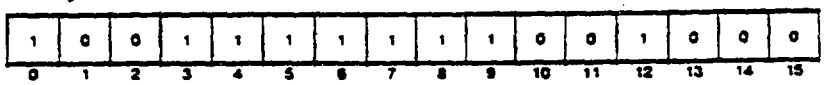
Effect: The words popped and their destinations are as follows:

Word Popped	Contents
1	Bit 0 = carry bit; bits 1-15 = PC
2	AC3
3	AC2
4	AC1
5	AC0

Continues operation with the word addressed by the updated program counter. The frame pointer remains unchanged.

Pop PC and Jump (POPJ)

POPJ



Summary

SP — PC

Detailed Description

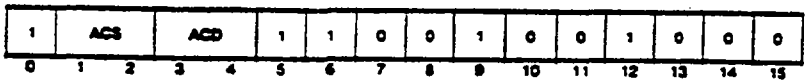
Use: To pop the top word off the stack and place it in the program counter.

Effect: Pops the top word off the stack and places it in the program counter. Continues execution with the word addressed by the updated value of the program counter.

Decrements the stack pointer by one. Leaves the frame pointer unchanged.

Push Multiple Accumulators (PSH)

PSH *acs,acd*



Summary

(AC0, AC1, AC2, or AC3) — SP (One to four ACs are pushed)

Detailed Description

Use: To push the contents of one to four accumulators onto the stack.

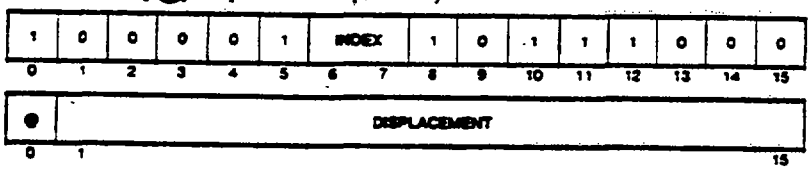
Effect: Pushes accumulators in ascending order starting with ACS and ending with ACD. AC3 + 1 is AC0. If you specify ACS and ACD as the same accumulator, the instruction pushes that accumulator onto the stack.

Increments the stack pointer by the number of accumulators pushed. Leaves the frame pointer unchanged.

Checks for overflow after completing the entire push operation.

Push Jump (PSHJ)

PSHJ *[@]displacement[,index]*



Summary

Effective address — PC

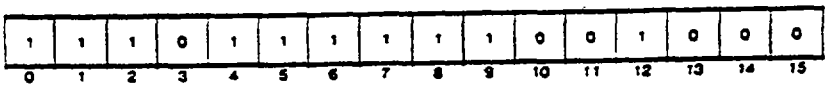
Detailed Description

Use: To push the next address onto the stack.

Effect: Pushes the address of the next sequential instruction onto the stack, computes the effective address, and places it in the program counter. It continues execution with the word addressed by the updated value of the program counter.

Restore (RSTR)

RSTR



Summary

9 words from the SP — predetermined locations

Detailed Description

Use: To pop nine words off the stack and place them in predetermined locations.

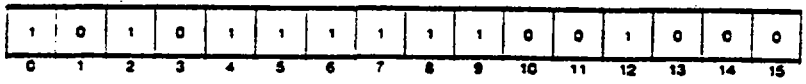
Effect: The words popped and their destinations are as follows:

Word Popped	Contents
1	Bit 0 = carry bit; bits 1-15 = PC
2	AC3
3	AC2
4	AC1
5	AC0
6	Stack fault pointer
7	Stack limit
8	Frame pointer
9	Stack pointer

Continues operation with the word addressed by the updated program counter.

Return (RTN)

RTN



Summary

SP (a return block) — popped

Detailed Description

Use: To pop a return block off the stack.

Effect: Returns control from subroutines that have issued a **SAVE** instruction at their entry points. The **SAVE** instruction loads the current value of the stack pointer into the frame pointer. The **RTN** instruction places the contents of the frame pointer into the stack pointer and pops five words from the stack.

Word Popped	Destination
1	Bit 0 = carry bit; bits 1-15 = PC
2	AC3
3	AC2
4	AC1
5	AC0

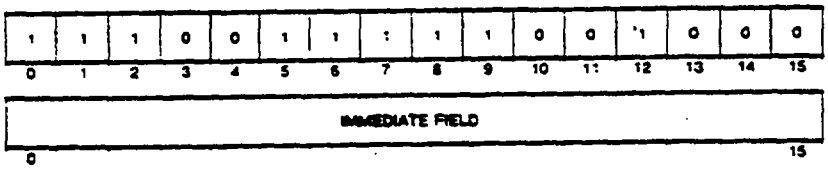
The operations occur as follows:

1. The contents of the frame pointer are loaded into the stack pointer.
2. Bit 0 of the location addressed by the new stack pointer is loaded into the carry and bits 1-15 are loaded into the program counter. The stack pointer decrements by one.
3. The contents of the location now addressed by the stack pointer are loaded into AC3. The stack pointer decrements by one.
4. The contents of the location now addressed by the stack pointer are loaded into AC2. The stack pointer decrements by one.
5. The contents of the location now addressed by the stack pointer are loaded into AC1. The stack pointer decrements by one.
6. The contents of the location now addressed by the stack pointer are loaded into AC0. The stack pointer decrements by one. The stack pointer now points to the new top of the stack.
7. The contents of AC3 are loaded into the frame pointer.

Continues operation with the word addressed by the updated program counter.

Save (SAVE)

SAVE *i*



Summary

Information needed by the return instruction — SP

Detailed Description

Use: To save the information required by the **RTN** instruction.

Effect: Pushes a return block onto the stack. Then, places the value of the incremented stack pointer in the frame pointer and in AC3. Leaves AC0, AC1, and AC2 unchanged. The following table shows the format of the 5-word return block:

Word Pushed	Contents
1	AC0
2	AC1
3	AC2
4	Frame pointer before the save
5	Bit 0 = carry bit; bits 1-15 = bits 1-15 of AC3

The operations occur as follows:

1. The SP increments by one. The contents of AC0 are written into the location now addresed by the stack pointer.
2. The SP increments by one. The contents of AC1 are written into the location now adressed by the stack pointer.
3. The SP increments by one. The contents of AC2 are written into the location now addressed by the stack pointer.
4. The SP increments by one. A zero is written into bit 0, and the contents of the frame pointer are written into bits 1-15 of the location now addressed by the SP.
5. The SP increments by one. The carry is written into bit 0, and bits 1 to 15 of AC3 are written into bits 1 to 15 of the location now addressed by the SP.
6. Checks for stack overflow. An overflow occurs if the SP exceeds the stack limit. If an overflow occurs, it pushes a stack fault block on the stack, overwriting the first return block.
7. If an overflow has not occurred, it puts the stack pointer in the frame pointer and in AC3.
8. If an overflow has not occurred, it adds the immediate field to the stack pointer.

The **SAVE** instruction allocates a part of the stack for use by the procedure that executed the **SAVE**. After storing the SP in the frame pointer and pushing the return block, **SAVE** adds the unsigned 16-bit integer in its immediate field to the SP. This unsigned integer is called the *frame size*.

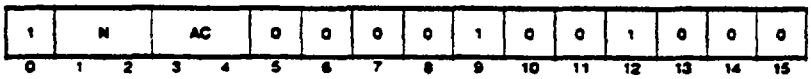
The frame size defines a portion of the stack not normally accessed by push and pop operations. The procedure issuing the **SAVE** instruction uses this area for temporary storage of variables, counters, etc. The frame pointer whose value goes into AC3 points to this storage area.

After pushing the 5-word return block, the **SAVE** instruction checks for stack overflow. If the contents of the SP exceed the contents of the stack limit, a stack overflow occurs.

Use the **SAVE** instruction after a **JSR** instruction. **JSR** transfers control to a subroutine and stores the return address in AC3. **SAVE** pushes AC3 on the stack, then puts the frame pointer in AC3.

Subtract Immediate (SBI)

SBI *n,ac*



Summary

(AC - N) → AC
(Where N is an unsigned integer from 1 to 4)

Detailed Description

Use: To subtract N, an unsigned integer from one to four, from the contents of an accumulator.

Effect: Subtracts the contents of the immediate field plus 1 from the unsigned, 16-bit number in the specified accumulator. This is done by adding 1 to the contents of the immediate field, forming its two's complement, and then adding the result to the contents of the accumulator. The carry bit remains unchanged.

Notes

- DGC assemblers compute N before loading the immediate field. You code *n*, which is N + 1. Code the exact value you want to subtract.

Skip if ACS Greater than or Equal to ACD (SGE)

SGE *acs,acd*

Summary

If ACS is >= ACD, then skip

Detailed Description

Use: To compare two signed integers in two accumulators and skip if the first is greater than or equal to the second.

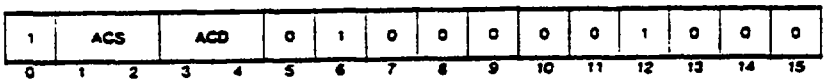
Effect: Algebraically compares the two's complement number in ACS to the two's complement number in ACD. Skips the next sequential word if the number in ACS is greater than or equal to the number in ACD. The contents of ACS and ACD remain unchanged.

Notes

- The SGT and SGE instructions treat the contents of the specified accumulators as signed, two's complement integers. To compare unsigned integers, use either the SUB or ADC instruction.

Skip if ACS Greater than ACD (SGT)

SGT *acs,acd*



Summary

If ACS is > ACD then skip

Detailed Description

Use: To check if one number is greater than another.

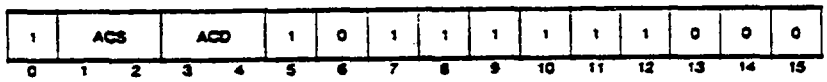
Effect: Algebraically compares the two's complement number in ACS to the two's complement number in ACD. Skips the next sequential word if the number in ACS is greater than the one in ACD. The contents of ACS and ACD remain unchanged.

Notes

- **SGT** treats the contents of the specified accumulators as two's complement integers. To compare unsigned integers, use either **SUB** or **ADC** and set the [skip] field appropriately.

Skip on Nonzero Bit (SNB)

SNB *acs,acd*



Summary

If bit = 1, skip the next word

Detailed Description

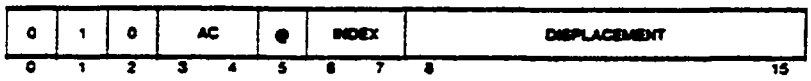
Use: To skip the next sequential word if the addresed bit is set to 1.

Effect: Forms a bit pointer from ACS and ACD. ACS contains the effective address. Bits 0-11 of ACD are the word offset, and bits 12-15 of ACD specify the bit. ACS and ACD remain unchanged.

If you specify ACS and ACD as the same accumulator, the effective address is zero and the accumulator contains the word offset and specifies the bit.

Store Accumulator (STA)

STA *ac,[@]displacement[,index]*



Summary

(AC) → memory location

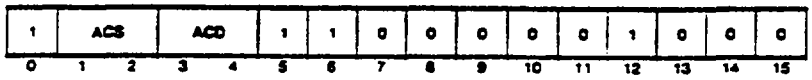
Detailed Description

Use: To store the contents of an accumulator into a memory location.

Effect: Places the contents of the specified accumulator in the word addressed by the effective address. Overwrites the previous contents of the addressed location. The contents of the specified accumulator remain unchanged.

Store Byte (STB)

STB *acs,acd*



Summary

(High byte of ACD) → stored in an addressed memory byte

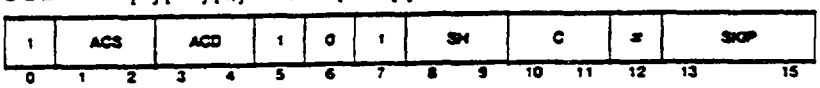
Detailed Description

Use: To store the high-order byte of an accumulator into memory.

Effect: Stores the high byte (bits 8 to 15) of ACD in an addressed memory byte. ACS contains the byte pointer. ACS and ACD remain unchanged.

Subtract (SUB)

SUB *[c][sh][#]acs,acd[,skip]*



Summary

ACD - ACS — ACD

Detailed Description

Use: To perform unsigned integer subtraction.

Effect: Sets carry to the specified value.

Subtracts the unsigned, 16-bit number in ACS from the unsigned, 16-bit number in ACD by taking the two's complement of the number in ACS and adding it to the number in ACD. If the operation produces a carry of 1 out of the most-significant bit (bit 0), the instruction complements the carry.

Places the 17-bit value (carry and the result of the subtraction) in the shifter. Performs the specified shift operation. If the no-load bit is zero, places the 17-bit value in the carry bit and ACD.

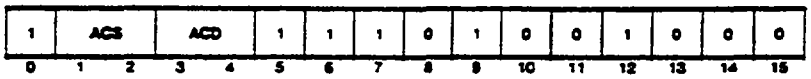
Tests the skip condition. If the condition is true, skips the next sequential word.

Notes

- If, before execution, the 16-bit, unsigned number in ACS is less than or equal to the 16-bit, unsigned number in ACD, the instruction complements the carry.

System Call (SYC)

SYC *acs,acd*



Summary

Return block is pushed; control transferred to system call handler

Detailed Description

Use: To push a return block and transfer control to the system call handler.

Effect: Disables the map, pushes the standard return block on the stack, and jumps indirectly through location 2. ACS and ACD remain unchanged.

The program counter in the return block contains the address of the instruction immediately following **SYC**.

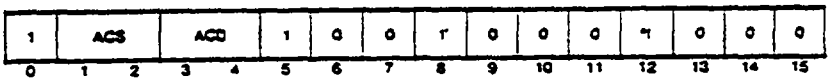
I/O interrupts cannot occur between the time the **SYC** instruction is executed and the time the next instruction is executed.

Notes

- If both accumulators are specified as AC0, the instruction does not push a return block onto the stack. AC0 remains unchanged.
- Data General Corporation assemblers recognize the mnemonic **SCL** as equivalent to **SYC** 1,1, and **SVC** as equivalent to **SYC** 0,0.

Skip on Zero Bit (SZB)

SZB *acs.acd*



Summary

If bit = 0, skip the next word

Detailed Description

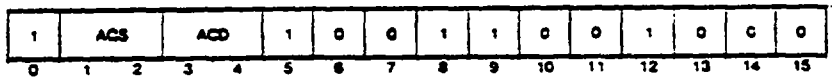
Use: If the addressed bit is zero, the next sequential word is skipped.

Effect: Forms a bit pointer from ACS and ACD. ACS contains the effective address. Bits 0-11 of ACD are the word offset, and bits 12-15 specify the bit. ACS and ACD remain unchanged.

If you specify ACS and ACD as the same accumulator, the effective address is zero. The accumulator contains the word offset and specifies the bit.

Skip on Zero Bit and Set to One (SZBO)

SZBO *acs.acd*



Summary

If the bit = 0, set that bit to 1, and skip the next word

Detailed Description

Use: If the addressed bit is zero, sets the bit to one and skips the next sequential word.

Effect: Forms a bit pointer from ACS and ACD. ACS contains the effective address. Bits 0-11 of ACD are the word offset, and bits 12-15 specify the bit. ACS and ACD remain unchanged.

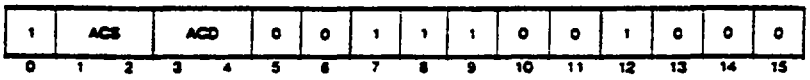
If you specify ACS and ACD as the same accumulator, the effective address is zero. The accumulator contains the word offset and specifies the bit. The CPU controls the memory bus until the instruction is completed.

Notes

- You can use this instruction to implement bit maps. Bit maps allocate facilities (such as memory blocks and I/O devices) to several processes or tasks that interrupt one another or that run in a multiprocessor environment.

Exchange Accumulators (XCH)

XCH *acs.acd*



Summary

AC —— AC (exchange accumulators)

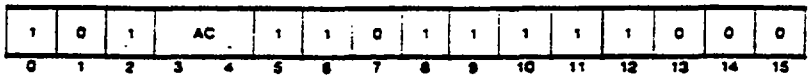
Detailed Description

Use: To exchange the contents of two accumulators.

Effect: Places the original contents of ACS in ACD and the original contents of ACD in ACS.

Execute (XCT)

XCT *ac*



Summary

Executes the instruction in AC

Detailed Description

Use: To execute the instruction contained in AC as if it were in main memory, in the location of the **XCT** instruction.

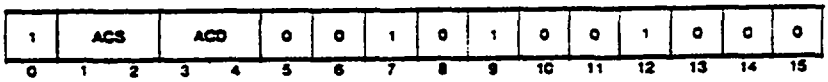
Effect: If the instruction in the AC is an **XCT** instruction that executes itself, the processor is placed in a 1-instruction loop. Because of this possibility, an I/O interrupt can occur just before the processor executes the instruction AC. If the I/O interrupt occurs, then the program counter in the return block is pushed onto the stack that contains the address of the **XCT** instruction. This instruction gives you an effective "Wait for I/O Interrupt" instruction.

Notes

- If the specified accumulator contains the first word of a 2-word instruction, the word following the **XCT** instruction is used as the second word. Normal sequential operation then continues from the second word after the **XCT** instruction.
- Do not use the **XCT** instruction to execute an instruction that requires all four accumulators (such as **CMV, CMT, CMP, CTR,** or **BAM**).

Exclusive OR (XOR)

XOR *acs,acd*



Summary

ACS +O ACD → ACD (+O is exclusive OR notation)

Detailed Description

Use: To form the logical exclusive OR of the contents of two accumulators.

Effect: Forms the logical exclusive OR of the contents of ACS and the contents of ACD. Places the result in ACD.

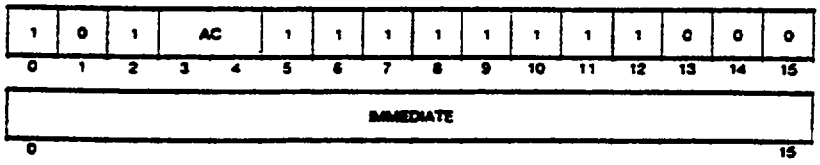
Sets a bit to one if the corresponding bit positions in the two operands are unlike. Otherwise, sets the bit to zero. The contents of ACS remain unchanged.

Example

Operation	Syntax	Before	After
Exclusive-ORs the contents of AC0 with the contents of AC1.	**XOR** 0,1	AC0 = 003156_8 AC1 = 020701_8	AC0 = 023657_8 AC1 = 020701_8

Exclusive OR Immediate (XORI)

XORI *i,ac*



Summary

i +O AC — AC (+O is exclusive OR notation)

Detailed Description

Use: To form the logical exclusive OR of an immediate field and an accumulator.

Effect: Forms the logical exclusive OR of the contents of the immediate field and the contents of the specified AC. Places the result in the specified AC.

Appendix B
Device and Interrupt Code Assignments

Table B.1 lists the I/O device code assignments for ILC local devices.

Table B-1. ILC Device Code Assignments

Device Code	Device
40	Reserved for Console Driver
50	System Control Port
30	Mover Device

All of the devices listed in Table B.1 can generate at least one interrupt to the microECLIPSE. Table B.2 lists the interrupt sources, the interrupt device code, and the order of priority of the interrupt for each source.

Table B-2. ILC Interrupt Information

Interrupt Device Code	Interrupt Source	Interrupt Priority
52	Power fail indication	1
56	Busy flag (host START)	2
51	Link device	3
57	Mover device	4
53	Real Time Clock (5.2 ms)	5
54	Real Time Clock (52 ms)	6
55	Real Time Clock (520 ms)	7
47	Reserved for console device	8

Appendix C
Signal Lists

This appendix lists signals for the new ILC internal cables, and for the standard host data channel pins.

The ILC uses two kinds of internal cables, one with compliant machines, the other with non-compliant machines. A compliant machine is one that complies with Federal Communications Commission (FCC) Electro-Magnetic Interference/Radio Frequency Interference (EMI/RFI) levels. The internal cable connecting the ILC Mem/IO slot to the EMI enclosure panel of a compliant CPU is 2.5 feet long. Connection to the EMI enclosure panel is through a 50-pin to 15-pin metal filler.

The internal cable connecting the ILC Mem/IO slot to the convenience panel for existing non-compliant machines is a 4-foot cable. The 4-foot cable connects to the convenience panel through a 15-pin to 25-pin adapter plate.

Both of these cables meet the standards described in Section 7.2, "Transceiver Cable Compatibility Interface Specifications" in the *Ethernet Data Link/Physical Layer Specification.*

Internal Cable Signal List

Table C.1 shows the location of the various I/O bus signals, and the power and ground lines on the backpanel.

Note that the shell connection that has designation PG in the table, is made at the EMI-tight enclosure end of the cable (DBa-15 connector end), and therefore no internal cable wire connection exists.

Table C-1. Internal Cable Signals

DBa Pin	Circuit Designation	Description	Backpanel Pin
03	DO-A	Data Out Circuit A	A-79
10	DO-B	Data Out Circuit B	A-77
11	DO-S	Data Out Shield	A-75
05	DI-A	Data In Circuit A	A-85
12	DI-B	Data In Circuit B	A-83
04	DI-S	Data In Shield	A-81
02	CI-A	Control In A	A-90
09	CI-B	Control In B	A-88
01	CI-S	Control In Shield	A-86
06	VC	Voltage Common	A-89
13	VP	Voltage Plus	A-91
14	VS	Voltage Shield	A-87
Shell	PG	Protective Ground	None

Standard Host/Data Channel Interface Signals

- The standard host Data Channel pin designations are listed in Table C.2.

  Note that the signals marked with an asterisk are part of the transceiver (XCVR) interface. Connection of these signals to the transceiver is via transceiver internal and external cables.

Table C-2. Host/Data Channel Interface Signals

Backpanel Pin Number	Signal Description	
A-01	GND	
A-02	GND	
A-03	+5V	
A-04	+5V	
A-38	-MSKO	
A-40	INTA	
A-46	-DS3	
A-47	EXT HST	
A-50	CLR	
A-52	STRT	
A-60	-DCHA	
A-62	-DS4	
A-64	-DS5	
A-66	-DS2	
A-68	-DS1	
A-70	IORST	
A-72	-DS0	
A-74	IOPLS	
A-75	GND	
A-77	-TRMT (TO XCVR)	*
A-79	TRMT (TO XCVR)	*
A-80	-SELD	
A-81	GND	
A-82	-SELB	
A-83	-RCV (FROM XCVR)	*
A-85	RCV (FROM XCVR)	*
A-86	GND	
A-87	GND (GND SH TO XCVR)	*
A-88	-CLSN (FROM XCVR)	*
A-89	GND (GND VC TO XCVR)	*
A-90	+CLSN (FROM XCVR)	*
A-91	+12V (TO XCVR)	*
A-92	GND	
A-93	-DCHPOUT	
A-94	-DCHPIN	
A-95	-INTPOUT	
A-96	-INTPIN	
A-97	+5V	
A-98	+5V	
A-99	GND	
A-100	GND	

(continued)

Table C-2. Host/Data Channel Interface Signals (continued)

Backpanel Pin Number	Signal Description
B-01	GND
B-02	GND
B-03	+5V
B-04	+5V
B-17	-DCHM0
B-21	-POWER FAIL
B-29	-INTR
B-33	DCHO
B-55	-DATA7
B-37	DCHI
B-39	POWER OK
B-41	-RQENB
B-50	GND
B-55	-DATA7
B-56	-DATA14
B-57	-DATA5
B-58	-DATA11
B-59	-DATA12
B-60	-DATA8
B-61	-DATA4
B-62	-DATA0
B-63	-DATA9
B-64	-DATA13
B-65	-DATA1
B-66	-DATA15
B-73	-DATA3
B-74	-XDCH
B-75	-DATA10
B-82	-DATA2
B-88	+12V
B-95	-DATA6
B-97	+5V
B-98	+5V
B-99	GND
B-100	GND

Appendix D
Sample Programs

This appendix contains examples of programs that provide communication between the host CPU and the ILC microECLIPSE, and that initialize and direct the Intel 82586 LANCC.

These are sample programs, and are not intended to encompass all possible situations. They are presented as a guide to programming the ILC.

microECLIPSE System Program Examples

This section contains examples of programs used to control the host/microECLIPSE interface. The examples given let you:

- Send 20 bits of data, plus a parity bit to the ILC from the host using serial protocol
- Issue a VIO command to the ILC
- Load a block of data into the ILC, and verify the data
- Load the ILC code module into the controller, and run the code module.

Sending Data and Parity Bits

```
; SCOM - SEND 20 BIT DATA PLUS PARITY VIA SERIAL PROTOCOL
;
;       calling sequence:
;               JSR     @.SCOM          ;send serial data
;               pktptr                  ;address of data packet
;               error return
;               normal return
;       packet format:
;               word 0 - high order 4 bits (right justified)
;               word 1 - low order 16 bits
SCOM:
        INC     3.3             ;POINT PAST PARAM PTR
        STA     3.SCOMS         ;SAVE AC'S
        STA     2.SCOMS+1
        STA     1.SCOMS+2
        STA     0.SCOMS+3
        LDA     3.-1.3          ;GET PTR TO DATA
        LDA     1.0.3           ;GET DATA (HIGH 4 BITS)
        LDA     2.1.3           ;         (LOW 16 BITS)

        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOC.0          ; OR IN NIOC COMMAND
        XCT     0               ;RESET DEVICE
```

```
        LDA     0.DEVICE        :GET DEVICE CODE
        IORI    NIOP.0          : OR IN NIOP COMMAND
        XCT     0               :START SERIAL PROTOCOL

        ELEF    3.12.           :LEFT JUSTIFY DATA
        NEG     3.3
SCOM1:                          :LOOP
        MOVZL   2.2             :  DOUBLE SHIFT AC1.AC2 LEFT
        MOVL    1.1
        INC     3.3.SZR         :  EXIT IF DONE
        JMP     SCOM1           :END_LOOP

        ELEF    3.20.           :INIT BIT COUNTER
        STA     3.SCOMC
        SUB     3.3             :INIT PARITY
        STA     3.SCOMD


SCOM2:                          :LOOP
        JSR     @IWTDN          :  WAIT FOR DONE SET
        JSR     SCOME           :   ** TIMEOUT ON DONE SET

        LDA     0.DEVICE        :GET DEVICE CODE
        IORI    SKPBZ.0         : OR IN SKPBZ COMMAND
        XCT     0               :  BUSY SHOULD BE RESET

        JSR     SCOME           :   ** BUSY NOT RESET
        MOVZL   2.2             :  DOUBLE SHIFT AC1.AC2 LEFT
        MOVL    1.1.SNC         :  IF "1" FALLS OUT THEN
        JMP     SCOM3

        LDA     0.DEVICE        :GET DEVICE CODE
        IORI    NIOS.0          : OR IN NIOS COMMAND
        XCT     0               :    ISSUE START TO SEND 1

        LDA     0.DEVICE        :GET DEVICE CODE
        IORI    SKPDN.0         : OR IN SKPDN COMMAND
        XCT     0               :    CHECK FOR BUSY SET

        JSR     SCOME           :    ** BUSY NOT SET AFTER NIOS
        ISZ     SCOMD           :    TOGGLE PARITY
SCOM3:                          :  END_IF
        LDA     0.DEVICE        :GET DEVICE CODE
        IORI    NIOP.0          : OR IN NIOP COMMAND
        XCT     0               :  ISSUE IOPLS TO "CLOCK" BIT
        DSZ     SCOMC           :  EXIT IF ALL BITS SENT
        JMP     SCOM2           :END_LOOP

        JSR     @IWTDN          :WAIT FOR DONE SET
        JSR     SCOME           : ** TIMEOUT ON DONE SET
        LDA     3.SCOMD         :GET PARITY
        MOVR#   3.3.SNC         :IF ODD # BITS SET THEN
        JMP     SCOM4
```

```
        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOS.0          ; OR IN NIOS COMMAND
        XCT     0               ;  SET PARITY BIT
SCOM4:                          ;END_IF

        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOP.0          ; OR IN NIOP COMMAND
        XCT     0               ;SEND PARITY BIT

        JSR     @INTDN          ;WAIT FOR DONE SET
        JSR     SCOME           ; ** PARITY ERROR

        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOS.0          ; OR IN NIOS COMMAND
        XCT     0               ;SEND STOP BIT

        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOP.0          ; OR IN NIOP COMMAND
        XCT     0

        JSR     @INTDN          ;WAIT FOR DONE SET
        JSR     SCOME           ; ** STOP BIT NOT ACKNOWLEDGED

        LDA     0.DEVICE        ;GET DEVICE CODE
        IORI    NIOP.0          ; OR IN NIOP COMMAND
        XCT     0               ;CLEAR DONE
        ISZ     SCOMS           ;BUMP FOR GOOD RETURN
        LDA     0.SCOMS+3       ;RESTORE AC'S
        LDA     1.SCOMS+2
        LDA     2.SCOMS+1
SCOME:
        STA     2.SCOMX         ;SAVE ERROR PC
        JMP     @SCOMS          ;RETURN

SCOMS:  .BLK    4               ;AC SAVE AREA
SCOMX:  0                       ;ERROR PC SAVE
SCOMC:  0                       ;BIT COUNTER
SCOMD:  0                       ;PARITY ACCUMULATOR
```

Sending a VIO Command to the ILC

```
; PCOM - ISSUE VIO COMMAND TO ILC
;
;       calling sequence:
;               JSR     @.PCOM          ;give prom command
;               pktptr                  ;ptr to PCOM packet
;               error return
;               normal return
;       packet format:
;               word 0 - VOA data
;               word 1 - VOB data
;               word 2 - VOC data
PCOM:
        INC     3,3             ;POINT PAST PARAM PTR
```

```
        STA     3,PCOMS         ;SAVE AC'S
        STA     2,PCOMS+1
        STA     1,PCOMS+2
        STA     0,PCOMS+3

        LDA     3,-1,3          ;INIT PTR TO COMMAND DATA
        LDA     1,.VOA,3        ;INIT VOA,VOB,VOC
        STA     1,VOA
        LDA     1,.VOB,3
        STA     1,VOB
        LDA     1,.VOC,3
        STA     1,VOC
        SUB     1,1             ;RESET VIA,VIB,VIC
        STA     1,VIA
        STA     1,VIB
        STA     1,VIC
        LDA     0,DEVICE        ;GET DEVICE CODE

        LDA     0,DEVICE        ;GET DEVICE CODE
        IORI    NIOS,0          ; OR IN NIOS COMMAND
        XCT     0               ;START LNC

        JSR     @INTDN          ;WAIT FOR DONE
        JSR     PCOME           ; ** TIMEOUT ON DONE
        LDA     1,VIC           ;CHECK VIC
        MOVL#   1,1,SZC
        JSR     PCOME           ; ** COMMAND FAILURE

        LDA     0,DEVICE        ;GET DEVICE CODE
        IORI    NIOP,0          ; OR IN NIOP COMMAND
        XCT     0               ;RESET DONE

        ISZ     PCOMS           ;BUMP FOR GOOD RETURN
        LDA     0,PCOMS+3       ;RESTORE AC'S
        LDA     1,PCOMS+2
        LDA     2,PCOMS+1
PCOME:
        STA     3,PCOMX         ;SAVE ERROR PC
        JMP     @PCOMS          ;RETURN

PCOMS:  .BLK    4               ;AC SAVE AREA
PCOMX:  0                       ;ERROR PC SAVE AREA
```

Loading and Verifying a Data Block

```
; LBLK - LOAD A BLOCK OF DATA INTO DEVICE AND VERIFY
;
;       calling sequence:
;               JSR @.LBLK              ;load block and check
;               pktptr                  ;packet ptr
;               error return
;               normal return
;       packet format:
;               word 0 - host address
```

```
;               word 1 - controller logical address
;               word 2 - word count
LBLK:
        INC     3.3             ;POINT PAST PARAM PTR
        STA     3.LBLKS         ;SAVE AC'S
        STA     2.LBLKS+1
        STA     1.LBLKS+2
        STA     0.LBLKS+3
;
; SETUP HOST ADDRESS
;
        LDA     3.-1.3          ;GET PACKET PTR
        LDA     2.LBLKA         ;INIT PTR TO PCOM PACKET
        LDA     0.0.3           ;VOB = HOST ADDRESS
        STA     0..VOB.2
        STA     0.LBLKJ         ;SAVE FOR COMPARE
        JSR     @.PCOM          ;GIVE COMMAND
        LBLKB
        JSR     LBLKE           ; ** COMMAND ERROR
;
; LOAD CODE BLOCK
;
        LDA     3.LBLKS         ;INIT PACKET PTR
        LDA     3.-1.3
        LDA     2.LBLKC         ;INIT PTR TO PCOM PACKET
        LDA     0.1.3           ;VOA = CONTROLLER ADDRESS
        STA     0..VOA.2
        LDA     0.2.3           ;VOB = WORD COUNT
        STA     0..VOB.2
        JSR     @.PCOM          ;GIVE COMMAND
        LBLKD
        JSR     LBLKE           ; ** COMMAND ERROR
;
; READBACK LOADED BLOCK AND COMPARE
;
        LDA     3.LBLKS         ;INIT PACKET PTR
        LDA     3.-1.3
        LDA     0.LBLKI         ;SET HOST ADDRESS
        LDA     2.LBLKA         ; IN 'SET HOST ADDR' PKT
        STA     0..VOB.2
        LDA     0.1.3           ;SET CONTROLLER ADDRESS
        LDA     2.LBLKF         ; IN 'DUMP BLOCK' PKT
        STA     0..VOA.2
        LDA     0.2.3           ;SETUP WORD COUNTER
        STA     0.LBLKH


LBLK2:                          ;LOOP
        JSR     @.PCOM          ;  GIVE 'SET HOST ADDR' CMND
        LBLKB
        JSR     LBLKE           ;    ** COMMAND FAILURE
        JSR     @.PCOM          ;  DUMP A WORD BACK TO HOST
        LBLKG
        JSR     LBLKE           ;    ** COMMAND FAILURE
        LDA     0.@LBLKI        ;  GET RETURNED WORD
```

```
        LDA     1,@LBLKJ        ;  GET ORIGINAL
        SUBZ#   0,1,SZR         ;  COMPARE
        JSR     LBLKE           ;    ** DATA COMPARE FAILED
        ISZ     LBLKJ           ;  UPDATE BUFFER PTR
        ISZ     .VOA.2          ;  UPDATE CONTROLLER ADDR
        DSZ     LBLKH           ;  EXIT IF ALL CHECKED
        JMP     LBLK2           ;END_LOOP

        ISZ     LBLKS           ;BUMP FOR GOOD RETURN
        LDA     0,LBLKS+3       ;RESTORE AC'S
        LDA     1,LBLKS+2
        LDA     2,LBLKS+1
LBLKE:
        STA     3,LBLKX         ;SAVE ERROR PC
        JMP     @LBLKS          ;RETURN
LBLKS:  .BLK    4               ;AC SAVE AREA
LBLKX:  0                       ;ERROR PC SAVE AREA
LBLKA:  LBLKB                   ;PTR TO SET HOST ADDR PKT
LBLKC:  LBLKD                   ;PTR TO LOAD BLOCK PKT
LBLKF:  LBLKG                   ;PTR TO DUMP BLOCK PKT
LBLKH:  0                       ;WORD COUNTER
LBLKI:  LBLKK                   ;PTR TO RETURN BUFFER
LBLKJ:  0                       ;PTR TO PACKET TO LOAD
LBLKB:                          ;'SET HOST ADDRESS' CMND PKT
        0                       ;  HIGH ORDER ADDRESS
        0                       ;  LOW ORDER ADDRESS
        5                       ;  COMMAND NUMBER
LBLKD:                          ;'LOAD BLOCK' COMMAND PKT
        0                       ;  CONTROLLER ADDRESS
        0                       ;  WORD COUNT
        11                      ;  COMMAND NUMBER
LBLKG:                          ;'DUMP BLOCK' COMMAND PKT
        0                       ;  CONTROLLER ADDRESS
        1                       ;  WORD COUNT
        6                       ;  COMMAND NUMBER
LBLKK:  .BLK    1               ;RETURN BLOCK BUFFER
```

Loading and Running ILC Code Module

```
; DLDV - LOAD ILC CODE MODULE INTO DEVICE AND RUN IT
;
;       calling sequence:
;               JSR @.DLDV              ;download and run
;               pktptr                  ;packet pointer
;               error return
;               normal return
;       packet format:
;               word 0 - host address
;               word 1 - controller logical address
;               word 2 - word count
DLDV:
        INC     3,3             ;POINT PAST PARAM PTR
        STA     3,DLDVS         ;SAVE AC'S
        STA     2,DLDVS+1
```

```
        STA     1,DLDVS+2
        STA     0,DLDVS+3
;
; LOAD CODE MODULE INTO ILC
;
        LDA     3,-1,3          ;INIT PARAM PTR
        STA     3,DLDVA         ; (SAME PKT FOR LBLK)
        JSR     @.LBLK          ;LOAD CODE BLOCK TO CONTROLLER
DLDVA:  0                       ; PACKET PTR
        JSR     DLDVE           ; ** LOAD BLOCK FAILURE
;
; BEGIN CODE MODULE EXECUTION
;
        LDA     3,DLDVS         ;INIT PARAM PTR
        LDA     3,-1,3
        LDA     0,1,3           ;INIT CONTROLLER ADDRESS
        LDA     2,DLDVB         ; IN 'RUN' CMND PKT
        STA     0,,VOA,2
        JSR     @.PCOM          ;ISSUE RUN COMMAND
        DLDVC
        JSR     DLDVE           ; ** COMMAND FAILURE

        ISZ     DLDVS           ;BUMP FOR GOOD RETURN
        LDA     0,DLDVS+3       ;RESTORE AC'S
        LDA     1,DLDVS+2
        LDA     2,DLDVS+1
DLDVE:
        STA     3,DLDVX         ;SAVE ERROR PC
        JMP     @DLDVS          ;RETURN

DLDVS:  .BLK    4               ;AC SAVE AREA
DLDVX:  0                       ;ERROR PC SAVE AREA
DLDVB:  DLDVC                   ;PTR TO RUN CMND PKT
DLDVC:                          ;'RUN' COMMAND PACKET
        0                       ;  CONTROLLER ADDRESS
        0                       ;
        3                       ;  COMMAND CODE


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; WTDN - WAIT FOR DONE SET WITH TIMEOUT
;       JSR @IWTDN
;       ERROR RETURN
;       NORMAL RETURN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
WTDN:
        STA     3,WTDNS         ;SAVE AC'S
        STA     2,WTDNS+1
        STA     0,WTDNS+2
        LDA     2,WTK1          ;INIT OUTER LOOP COUNT
WTDN1:                          ;LOOP
        LDA     0,DEVICE        ;GET DEVICE CODE
        IORI    SKPDZ,0         ; OR IN SKPDZ COMMAND
        XCT     0               ;  TAKE NORMAL RETURN IF DONE SET
```

```
        JMP     WTDN3
        SUBZL   0,0             ;  WAIT ABOUT A MILISECOND
        SCALL   DELAY?
        INC     2,2,SZR         ;  EXIT IF TIMEOUT
        JMP     WTDN1           ;END LOOP
        JMP     @WTDNS          ;ERROR RETURN
WTDN3:
        LDA     0,WTDNS+2       ;RESTORE AC'S
        LDA     2,WTDNS+1
        LDA     3,WTDNS         ;NORMAL RETURN
        JMP     1,3

WTDNS:  .BLK    3               ;AC SAVE AREA
WTK1:   -2000.                  ;ABOUT 2 SECONDS
```

Intel 82586 LANCC Program Examples

The first three parts of this section provide programs examples of:

- LANCC intialization
- Transmission and reception of data packets
- Interrupt handler routines

The last part of this section, titled LCC Equates, defines constants that are used in the three program examples.

LANCC Initialization

```
        ;
        ; ILCC - INIT LCC
        ;
        ;       PERFORM LCC INIT SEQUENCE
        ;       CONFIGURE LCC
        ;       SET UNIQUE ADDRESS
        ;       RESET ESI LOOPBACK BIT
        ;
        ;       RETURN TO CALL+1 ON ERROR
        ;       RETURN TO CALL+2 IF NO ERRORS
        ILCC:
                STA     3,ILCCS         ;SAVE AC'S
                STA     2,ILCCS+1
                STA     1,ILCCS+2
                STA     0,ILCCS+3
;
; SETUP SCP IN BANK 0 (TOP OF PHYSICAL MEMORY)
;
                IORST                   ;RESET LANCC
                ELEF    2,,SCP,0        ;ADDRESS OF SCP
                SUB     0,0             ;INIT TO 16 BIT BUS
                STA     0,SCP.SBS,2
                STA     0,SCP.ISP+1,2   ;HIGH BYTE OF ISCP PTR=0
                ELEF    0,,ISCP,0       ;GET ISCP POINTER
```

```
                MOVZL   0.0             ;MAKE BYTE PTR
                STA     0.SCP.ISP.2     ;SAVE LOW PART
;
; SETUP ISCP IN BANK B
;
                ELEF    2..ISCP.0       ;ADDRESS OF ISCP
                SUBZL   0.0
                STA     0.ISP.BZY.2     ; SET INITIALIZATION IN PROGRESS
                ELEF    0..SCB.0        ;ISP.SCO=SCB OFFSET
                MOVZL   0.0             ; (X 2 FOR BYTE ADDR)
                STA     0.ISP.SCO.2
                SUB     0.0             ;ISP.SCB=0 (SCB BASE)
                STA     0.ISP.SCB.2
                STA     0.ISP.SCB+1.2
;
; SETUP SCB IN BANK B
;
                ELEF    2..SCB.0        ;ADDRESS OF SCB
                SUB     0.0             ; CLEAR
                STA     0.SCB.STA.2     ;  STATUS WORD
                STA     0.SCB.CMD.2     ;  COMMAND WORD
;
; DISABLE LCC INTR. ISSUE CHANNEL ATTENTION
;
                ELDA    0.SCP.DOA       ;GET COPY OF SCP DOA
                IORI    SCP.MLK.0       ;MASK OUT LCC INTR TO POLL INTS
                ESTA    0.SCP.DOA       ;SAVE UPDATED COPY
                IORI    SCP.LKS.0       ;START LCC (CHANNEL ATTENTION)
                DOA     0.SCP           ;ISSUE COMMAND

        INITWT: DIA     0.SCP
                ANDI    SCP.LKD.0       ; IS LANCC INTERRUPT
                MOV     0.0.SNR         ;   SET?
                JMP     INITWT          ; NOT SET -- LOOP

                EJSR    LCCIH           ; ACK AND SAVE LANCC INTERRUPT BITS
                JMP     ILCCX           ; ** FAILED

                ELDA    0.SCP.DOA       ;CLEAR LANCC INTR
                IORI    SCP.LKC.0
                DOA     0.SCP



;
; CONFIGURE LCC
;
                EJSR    LCMD            ; CONFIGURE LCC
                ILCCA-.                 ; OFFSET TO CMD BLOCK
                JMP     ILCCX           ; ** COMMAND FAILED
;
; SET UNIQUE ADDR
;
                ELDA    0..UA4.0        ; BYTE 4 OF UNIQUE ADDR
                MOVS    0.0
                ELEF    2.ILCCF+1
```

```
        MOVZL   2,2
        STB     2,0             ; STORE IN CMD BLOCK
        ELDA    0,.UA56,0       ; STORE BYTES
        STA     0,ILCCF+2       ;   5 AND 6

        EJSR    LCMD            ; SET UNIQUE ADDR
        ILCCB-.                 ; OFFSET TO CMD BLOCK
        JMP     ILCCX           ; ** COMMAND FAILED

ILCCE:
        ISZ     ILCCS           ;BUMP FOR GOOD RETURN
        LDA     0,ILCCS+3       ;RESTORE AC'S
        LDA     1,ILCCS+2
        LDA     2,ILCCS+1

ILCCX:
        JMP     @ILCCS          ;RETURN

ILCCS:  .BLK    4               ;AC SAVE AREA

ILCCA:                          ;CONFIGURE CB
        0                       ;  STATUS
        CB.CCF                  ;  COMMAND
        0                       ;  LINK

ILCCFP: CFC.POD                 ;  CONFIGURE PARAMETERS
        CFC.P1D+CFC.SBF         ;   SAVE BAD FRAMES
        CFC.P2D
        CFC.P3D
        CFC.P4D
        CFC.P5D

ILCCB:                          ;SET UNIQUE ADDRESS CB
        0                       ;  STATUS
        CB.CAS                  ;  COMMAND
        0                       ;  LINK WORD

ILCCF:  10                      ;  08-00-
        33                      ;  18-XX-
        0                       ;  XX-XX


; LCMD - LCC COMMAND SUBROUTINE
;
;       CALLING SEQUENCE:
;               JSR     @.LCMD
;               pktptr-.
;               error return
;               normal return
;       PACKET FORMAT:
;               standard LCC action command block

LCMD:
        STA     2,LCMDS+1       ;SAVE AC'S
        STA     1,LCMDS+2
```

```
        STA     0,LCMDS+3

        LDA     2,0,3           ;GET PKT POINTER OFFSET
        ADD     3,2             ;COMPUTE EFFECTIVE ADDR
        INC     3,3             ;POINT PAST PARAM PTR
        STA     3,LCMDS         ;SAVE RETURN ADDR
        ELEF    1,16,,0         ;MOVE TO BANK B
        ELEF    3,,CB,0
        BLM
        ELEF    3,,CB,0         ;RESTORE CB POINTER
        SUB     0,0             ;CLEAR COMMAND BLOCK STATUS
        STA     0,CB.STA,3
        LDA     0,CB.CMD,3      ;SET EL,I
        IORI    CB.EL+CB.I,0
        STA     0,CB.CMD,3
        ELEF    2,,SCB,0        ;SCB ADDRESS
        SUB     0,0             ;
        IORI    SCB.CST,0       ;PUT START COMMAND UNIT
        STA     0,SCB.CMD,2     ;    CMD IN SCB
        MOVZL   3,0             ;SET SCB CBL PTR TO CB
        STA     0,SCB.CBL,2     ; (BYTE ADDRESS)
        ELDA    0,SCP,DOA       ;ISSUE CHANNEL ATTENTION
        IORI    SCP.LKS,0
        DOA     0,SCP

LCMD1:  BIA     0,SCP           ; LOOP
        ANDI    SCP.LKD,0       ; IS LANCC INTERRUPT
        MOV     0,0,SNR         ;   SET?
        JMP     LCMD1           ; NOT SET -- LOOP

        EJSR    LCCIN           ;  ACK AND SAVE INTERRUPT BITS
        JSR     LCMDE           ;  ** LCC INTR ERROR

        ELDA    0,LCCIS         ;  EXIT IF CX, CNR SET
        SUB     1,1
        IORI    1,SCB.SCX+SCB.SCN
        COM     1,1             ;RESET CX, CNR STATUS
        AND     1,0             ;   IN SAVE WORD
        ESTA    0,LCCIS

LCMD4:
        ELEF    3,,CB,0         ;INIT CB POINTER
        LDA     0,CB.STA,3      ;GET CB STATUS
        ANDI    CB.C+CB.B+CB.OK+CB.AB,0
        SUB     1,1
        IORI    1,CB.C+CB.OK    ;CHECK C,OK=1,B,AB=0
        SUB#    0,1,SZR
        JSR     LCMDE           ; ** BAD CB STATUS

        ISZ     LCMDS           ;BUMP FOR GOOD RETURN
        LDA     0,LCMDS+3       ;RESTORE AC'S
        LDA     1,LCMDS+2
        LDA     2,LCMDS+1

LCMDE:
```

```
            JMP     @LCMDS          ;RETURN

LCMDS:      .BLK    4               ;AC SAVE AREA
```

Packet Transmission

```
        ; TXPKT - TRANSMIT A PACKET
        ;
        ;       CALLING SEQUENCE:
        ;               JSR     @.TXPKT
        ;               PKTPTR-.
        ;               error return
        ;               normal return
        ;       PACKET FORMAT:
        ;               status word                     (0)
        ;               control word                    (1)
        ;               destination address (3 words)   (2-4)
        ;               type field                      (5)
        ;               buffer size (bytes)             (6)
        ;               buffer pointer                  (7)
        TXPKT:
                STA     3,TXPKTS        ;SAVE AC'S
                STA     2,TXPKTS+1
                STA     1,TXPKTS+2
                STA     0,TXPKTS+3
                LDA     2,0,3           ;COMPUTE PARAMETER POINTER
                ADD     2,3
                STA     3,TXPKTA        ;SAVE IT
;
; setup TBD
;
                 ELEF   2,TXPKTC        ;INIT TBD POINTER
                 LDA    0,6,3           ;INIT TBD BUFFER SIZE
                 IORI   TBD.EOF,0       ;SET END OF FRAME
                 STA    0,TBD.CNT,2
                 ELEF   3,,TBD,0        ;MOVE TO BANK B
                 ELEF   1,4,0
                 BLM
;
; setup TBUF
;
                 LDA    3,TXPKTA        ;INIT PARAM PTR
                 LDA    2,7,3           ;INIT BUFFER POINTER
                 LDA    1,6,3           ;INIT BYTE COUNT
                 INCZR  1,1             ;TRANSLATE TO WORD COUNT
                 ELEF   3,,TBUF,0       ;MOVE TO BANK B
                 BLM
;
; setup transmit CB
;
                 LDA    2,TXPKTA        ;INIT PARAM PTR
                 ADDI   2,2             ;POINT TO DEST ADDR
                 ELEF   3,TXPKTB+4      ;MOVE INFO TO TX CB
                 ELEF   1,4,0           ; (DEST ADDR, TYPE FIELD)
```

```
                BLM
;
; issue transmit command
;
                EJSR    LCMD            :ISSUE 'TRANSMIT' COMMAND
                TXPKTB-.
                JSR     TXPKTE          ; ** TRANSMIT FAILURE



;
; report tx status
;
                ELEF    3..CB.0         :INIT CB POINTER
                LDA     0.CB.STA.3      :GET STATUS WORD
                LDA     3.TXPKTA        :PUT STATUS WORD IN PARAM PKT
                STA     0.0.3

                ISZ     TXPKTS          :BUMP FOR GOOD RETURN
                LDA     0.TXPKTS+3      :RESTORE AC'S
                LDA     1.TXPKTS+2
                LDA     2.TXPKTS+1
        TXPKTE:
                LDA     3.TXPKTS        :RETURN PAST PARAM
                JMP     1.3

        TXPKTS: .BLK    4               :AC SAVE AREA

        TXPKTA: 0                       :PARAM PTR SAVE

        TXPKTB:                         :'TRANSMIT' COMMAND BLOCK (TXCB)
                0                       :  STATUS WORD
                CB.CTX                  :  COMMAND WORD (TRANSMIT)
                0                       :  LINK WORD
                .TBD*2                  :  TBD POINTER
                .BLK    3               :  DEST ADDR
                0                       :  TYPE FIELD

        TXPKTC:                         :TRANSMIT BUFFER DESCRIPTOR (TBD)
                0                       :  COUNT
                0                       :  LINK
                .TBUF*2                 :  BUFFER ADDRESS
                0
        ;
        ; STRCV - START LCC RECEIVE UNIT
        ;
        ;       CALLING SEQUENCE:
        ;               JSR     0.STRCV
        ;               error return
        ;               normal return
        STRCV:
                STA     3.STRCVS        :SAVE AC'S
                STA     2.STRCVS+1
                STA     1.STRCVS+2
                STA     0.STRCVS+3
```

```
        ELEF    2.STRCVA        ;COPY RFD TO BANK B
        ELEF    3..RFD.0
        ELEF    1.4.0
        BLM
        ELEF    2.STRCVB        ;COPY RBD TO BANK B
        ELEF    3..RBD.0
        ELEF    1.5.0
        BLM
        ELEF    3..SCB.0        ;POINT TO SCB
        ELEF    0..RFD.0        ;SETUP RFD PTR IN SCB
        MOVZL   0.0             ;TRANSLATE TO BYTE PTR
        STA     0.SCB.RFA.3
        SUB     0.0
        IORI    0.SCB.RST       ;START RU
        STA     0.SCB.CMD.3
        ELDA    0.SCP.DOA       ;ISSUE CHANNEL ATTENTION
        IORI    SCP.LKS.0
        DOA     0.SCP

        ISZ     STRCVS          ;BUMP FOR GOOD RETURN
        LDA     0.STRCVS+3      ;RESTORE AC'S
        LDA     1.STRCVS+2
        LDA     2.STRCVS+1


STRCVE:
        JMP     @STRCVS         ;RETURN

STRCVS: .BLK    4               ;AC SAVE AREA

STRCVA:                         ;RFD
        0                       ;  STATUS WORD
        RFD.EL                  ;  COMMAND WORD
        0                       ;  LINK WORD
        .RBD*2                  ;  RBD POINTER

STRCVB:                         ;RBD
        0                       ;  STATUS WORD
        0                       ;  LINK
        .RBUF*2                 ;  R-BUF POINTER
        0
        RBD.EL+40.              ;  EL. 40 BYTES

;
; WTPKT - WAIT FOR PACKET RECEIVED WITH TIMEOUT
;
;       CALLING SEQUENCE:
;               EJSR    WTPKT
;               timeout return
;               normal return
WTPKT:
        STA     3.WTPKTS        ;SAVE AC'S
        STA     2.WTPKTS+1
        STA     1.WTPKTS+2
        STA     0.WTPKTS+3
```

```
WTPKT1:                              ;LOOP
        ELDA    0,LCCIS              ;  GET UNSERVICED LCC INTR'S
        SUB     1,1
        IORI    1,SCB.SFR+SCB.SRN
        AND     1,0                  ;  EXIT IF FRAME IN, RU NOT RDY
        SUBZ#   1,0,SNR
        JMP     WTPKT2               ; FRAME IS HERE

WAITRC: DIA     0,SCP                ; LOOP
        ANDI    SCP.LKD,0            ; IS LANCC INTERRUPT
        MOV     0,0,SNR              ;   SET?
        JMP     WAITRC               ; NOT SET -- LOOP

        EJSR    LCCIH                ;  HANDLE THE INTERRUPT
        JSR     WTPKTE               ;   ** LCC INTR ERROR
        JMP     WTPKT1               ;END LOOP  -- CHECK IF ITS FR.RNR

WTPKT2:
        COM     1,1                  ;RESET FR. RNR STATUS BITS
        ELDA    0,LCCIS
        AND     1,0
        ESTA    0,LCCIS
        ISZ     WTPKTS               ;BUMP FOR GOOD RETURN
        LDA     0,WTPKTS+3           ;RESTORE AC'S
        LDA     1,WTPKTS+2
        LDA     2,WTPKTS+1
WTPKTE:
        JMP     @WTPKTS              ;RETURN

WTPKTS: .BLK    4                    ;AC SAVE AREA
```

Handling Interrupts

```
;
; 82586 INTERRUPT HANDLER
;
;       ACKNOWLEDGES AND RECORDS THE CAUSE OF LCC INTERRUPTS
;       NEW INTERRUPTS SET BITS IN VARIABLE "LCCIS"
;
;       CALLING SEQUENCE:
;               EJSR    LCCIH
;               error return
;               normal return
LCCIH:
        STA     3,LCCIHS             ;SAVE AC'S
        ELEF    2,,SCB,0             ;INIT PTR TO SCB
LCCIH1:                              ;LOOP
        LDA     1,SCB.CMD,2          ;  EXIT IF COMMAND WORD = 0
        MOV     1,1,SNR
        JMP     LCCIH2
        JMP     LCCIH1               ;END LOOP
LCCIH2:
        LDA     0,SCB.STA,2          ;GET STATUS
```

```
        SUB     1,1
        IORI    1,SCB.SCX+SCB.SFR+SCB.SCN+SCB.SRN
        AND     0,1,SNR         ;MASK ALL BUT INTR BITS
        JSR     LCCIHE          ; ** NO CAUSE FOR INTR
        LDA     0,LCCIS         ;GET OLD INTR STATUS
        IOR     1,0             ;INCLUDE NEW INTR'S
        STA     0,LCCIS         ;SAVE INTERRUPT STATUS
        STA     1,SCB.CMD,2     ;ACK INTR'S; NO-OP CU,RU
        ELDA    0,SCP,DOA       ;ISSUE CA; CLEAR LIR
        IORI    SCP.LKS+SCP.LKC,0
        DOA     0,SCP
        ISZ     LCCIHS          ;BUMP FOR GOOD RETURN
LCCIHE:
        JMP     @LCCIHS         ;RETURN
LCCIHS: .BLK    1
LCCIS:  0                       ;UNSERVICED LCC INTERRUPTS
```

LANCC Equates

```
        ; LCC (82586) EQUATES
        ;
        ;
        ; LCC SYSTEM CONFIGURATION POINTER (SCP) FORMAT
        ;

        SCP.SBS=0       ;SCP SYSTEM BUS CONTROL WORD
                        ;   0 = 16 BIT BUS
                        ;   1 = 8 BIT BUS
        SCP.ISP=3       ;SCP ISCP ADDRESS (2 WORDS)
                        ;   LOW ORDER 16 BITS
                        ;   HIGH ORDER 8 BITS
        SCP.CNT=5

;
; LCC INTERMEDIATE SYSTEM CONTROL POINTER (ISCP) FORMAT
;

        ISP.BZY=0       ;ISP BUSY
                        ;   1 = INITIALIZATION IN PROGRESS
                        ;   0 = SCB BASE AND OFFSET READ
        ISP.SCO=1       ;ISP SCB OFFSET
        ISP.SCB=2       ;ISP SCB BASE (2 WORDS)
                        ;   LOW ORDER 16 BITS
                        ;   HIGH ORDER 8 BITS
        ISP.CNT=4

        ;
        ; LCC SYSTEM CONTORL BLOCK (SCB) FORMAT
        ;

        SCB.STA=0       ;SCB STATUS WORD
         SCB.SCX=100000         ; BIT 0  - (CX) COMMAND EXECUTED
         SCB.SFR=040000         ; BIT 1  - (FR) FRAME RECIEVED
         SCB.SCN=020000         ; BIT 2  - (CNR) COMMAND UNIT NOT READY
```

```
SCB.SRN=010000          : BIT 3  - (RNR) RECEIVE UNIT NOT READY
                        : BIT 4  - 0
SCB.CSW=003400          : BITS 5-7 - (CUS) COMMAND UNIT STATUS
SCB.CUI=000000          :               0 - IDLE
SCB.CUS=000400          :               1 - SUSPENDED
SCB.CUR=001000          :               2 - READY
                        :               3-7 - NOT USED
                        : BIT 8  - 0
SCB.RSW=000160          : BITS 9-11 - (RUS) RECEIVE UNIT STATUS
SCB.RUI=000000          :               0 - IDLE
SCB.RUS=000020          :               1 - SUSPENDED
SCB.RUN=000040          :               2 - NO RESOURCES
                        :               3 - NOT USED
SCB.RUR=000100          :               4 - READY
                        :               5-7 - NOT USED
                        : BITS 12-15 - 0


SCB.CMD=1       :SCB COMMAND WORD
 SCB.ACX=100000         : BIT 0  - (ACK-CX) ACKNOWLEDGE CX
 SCB.AFR=040000         : BIT 1  - (ACK-FR) ACKNOWLEDGE FR
 SCB.ACN=020000         : BIT 2  - (ACK-CNR) ACKNOWLEGE CNR
 SCB.ARN=010000         : BIT 3  - (ACK-RNR) ACKNOWLEDGE RNR
                        : BIT 4  - NOT USED
                        : BITS 5-7 - (CUC) COMMAND UNIT COMMAND
 SCB.CNO=000000         :               0 - NOP
 SCB.CST=000400         :               1 - START CBL EXECUTION
 SCB.CRE=001000         :               2 - RESUME
 SCB.CSU=001400         :               3 - SUSPEND
 SCB.CAB=002000         :               4 - ABORT IMMEDIATELY
                        :               5-7 - ILLEGAL
 SCB.RES=000200         : BIT 8  - (RESET) HARDWARE RESET
                        : BITS 9-11 - (RUC) RECEIVE UNIT COMMAND
 SCB.RNO=000000         :               0 - NOP
 SCB.RST=000020         :               1 - START RECEIVING
 SCB.RRE=000040         :               2 - RESUME
 SCB.RSU=000060         :               3 - SUSPEND
 SCB.RAB=000100         :               4 - ABORT IMMEDIATELY
                        :               5-7 - ILLEGAL
                        : BITS 12-15 - NOT USED
SCB.CBL=2               :SCB COMMAND BLOCK LIST OFFSET
SCB.RFA=3               :SCB RECEIVE FRAME AREA OFFSET
SCB.CRC=4               :SCB # FRAMES DROPPED DUE TO CRC ERROR
SCB.ALN=5               :SCB # FRAMES DROPPED DUE TO ALIGNMENT ERROR
SCB.RSC=6               :SCB # FRAMES DROPPED DUE TO NO RESOURCES
SCB.OVR=7               :SCB # FRAMES LOST DUE TO BUS INAVAILABLE
SCB.CNT=8.


;
; LCC COMMAND BLOCK (CB) FORMAT
;
```

```
CB.STA=0        :CB STATUS WORD
 CB.C=100000    .       : BIT 0  - (C) COMMAND DONE
 CB.B=040000            : BIT 1  - (B) COMMAND BUSY
 CB.OK=020000           : BIT 2  - (OK) COMMAND EX OK
 CB.AB=010000           : BIT 3  - (AB) COMMAND ABORTED
                        : BITS 4-5 - COMMAND SPECIFIC STATUS
CB.CMD=1        :CB COMMAND WORD
 CB.EL=100000           : BIT 0  - (EL) END OF COMMAND LIST
 CB.S=040000            : BIT 1  - (S) SUSPEND AFTER THIS COMMAND
 CB.I=020000            : BIT 2  - (I) INTERRUPT AFTER THIS COMMAND
                        : BITS 3-12 - UNUSED
                        : BITS 13-15 - (CMD) COMMAND OP CODE
 CB.CN0=000000          :               0 - NOP
 CB.CAS=000001          :               1 - ADDRESS SETUP
 CB.CCF=000002          :               2 - CONFIGURE
 CB.CMS=000003          :               3 - MULTICAST ADDR SETUP
 CB.CTX=000004          :               4 - TRANSMIT
 CB.CTD=000005          :               5 - TDR
 CB.CDS=000006          :               6 - DUMP STATUS
 CB.CDI=000007          :               7 - DIAGNOSE
CB.LNK=2                :CB LINK TO NEXT CB

:
: LCC CONFIGURE COMMAND BLOCK FORMAT
:

                        :STATUS WORD
                        :COMMAND WORD
                        :OFFSET
 CFC.PW0=3              :PARAM WORD 0
                        : 0-3 - UNUSED
                        : 4-7 - FIFO LIMIT
                        : 8-11 - UNUSED
                        : 12-15 - BYTE CNT
 CFC.P0D=004014         : DEFAULT
                        :  FIFO_LIMIT=16.
                        :  BYTE_CNT=12.
 CFC.PW1=4              :PARAM WORD 1
  CFC.ELP=100000        : 0 - EXTERNAL LOOPBACK
  CFC.ILP=040000        : 1 - INTERNAL LOOPBACK
                        : 2-3 - PREAMBLE LENGTH
  CFC.ATP=004000        : 4 - ADDR/TYPE FIELD IN FRAME
                        : 5-7 - ADDR LEN
  CFC.SBF=000200        : 8 - SAVE BAD FRAMES
  CFC.SRD=000100        : 9 - EXTERNAL READY SYNC
                        : 10-15 - UNUSED
  CFC.P1D=023000        : DEFAULT
                        :  PREAMBLE_LENGTH = 8.
                        :  ADR_LEN = 6.
  CFC.PW2=5             :PARAM WORD 2
                        : 0-7 - INTERFRAME SPACING
                        : 8 - ALTERNATE BACKOFF METHOD
                        : 9-11 - EXPONENTIAL PRIORITY
                        : 12 - UNUSED
                        : 13-15 - LINEAR PRIORITY
```

```
          CFC.P2D=140000          ; DEFAULT
                                  ;  INTERFRAME_SPACING=96.
          CFC.PW3=6         ;PARAM WORD 3
                                  ; 0-3 - MAX RETRIES
                                  ; 4 - UNUSED
                                  ; 5-15 - SLOT TIME
          CFC.P3D=171000          ; DEFAULT
                                  ;  MAX_RETRIES = 15.
                                  ;  SLOT TIME = 512.
          CFC.PW4=7         ;PARAM WORD 4
                                  ; 0 - INTERNAL COLLISION DETECT
                                  ; 1-3 - CDT FILTER
                                  ; 4 - INTERAL CARRIER SENSE
                                  ; 5-7 - CRS FILTER
                                  ; 8 - PAD SLOT TIME
                                  ; 9 - BITSTUFFING
                                  ; 10 - ALTERNATE CRC (16 BIT)
                                  ; 11 - NO CRC INSERTION
                                  ; 12 - TRANSMIT ON NO CARRIER SENSE
                                  ; 13 - INTERNAL MANCHESTER CODING
          CFC.BCD=000002          ; 14 - BROADCAST DISABLE
          CFC.PRM=000001          ; 15 - PROMISCUOUS MODE
          CFC.P4D=000000          ; DEFAULT
          CFC.PW5=10        ;PARAM WORD 5
                                  ; 0-7 - UNUSED
                                  ; 8-15 - MINIMUM FRAME LENGTH
          CFC.P5D=000100          ; DEFAULT
                          ;  MIN_FRAME_LENGTH = 64.


;
; LCC TRANSMIT COMMAND BLOCK STATUS WORD FORMAT
;
                                  ; BIT 0 - (C) COMPLETED
                                  ; BIT 1 - (B) BUSY
                                  ; BIT 2 - (OK) NO ERRORS
                                  ; BIT 3 - (A) ABORTED
          TXC.NCS=004000          ; BIT 4 - NO CARRIER SENSE DURING TX
          TXC.NCT=002000          ; BIT 5 - LOSS OF CLEAR TO SEND
          TXC.UNR=001000          ; BIT 6 - DMA UNDERRUN
          TXC.DEF=000400          ; BIT 7 - TX DEFERRED
          TXC.HBT=000200          ; BIT 8 - HEARTBEAT
          TXC.TMC=000100          ; BIT 9 - TOO MANY COLLISIONS
                                  ; BIT 10 -
                                  ; BITS 11-15 - NUMBER OF COLLISIONS

;
; LCC TRANSMIT BUFFER DESCRIPTOR (TBD) FORMAT
;
          TBD.CNT=0         ;TBD COUNT
          TBD.EOF=100000          ; BIT 0  - (EOF) LAST BUFFER FOR THIS FRAME
                                  ; BITS 1-15 - NUMBER OF BYTES IN THIS BUFFER
          TBC.LNK=1         ;TBD LINK TO NEXT TBD
          TBD.BUF=2         ;TBD BUFFER ADDR (TWO WORDS .LOW FIRST)
;
```

```
; LCC RECEIVE FRAME DESCRIPTOR (RFD) FORMAT
;
        RFD.STA=0       :RFD STATUS
         RFD.C=100000           ; BIT 0 - FRAME RECEPTION COMPLETE (C)
         RFD.B=040000           ; BIT 1 - FRAME RECEPTIION IN PROGRESS (B)
         RFD.OK=020000          ; BIT 2 - FRAME RECEIVED WITHOUT ERRORS
                                ; BIT 3 -  RESERVED
         RFD.CRC=004000         ; BIT 4 - CRC ERROR (FRAME ALIGNED)
         RFD.ALG=002000         ; BIT 5 - CRC ERROR (FRAME MIS-ALIGNED)
         RFD.NBF=001000         ; BIT 6 - RAN OUT OF BUFFER SPACE
         RFD.OVR=000400         ; BIT 7 - DMA OVERRUN
         RFD.SHF=000200         ; BIT 8 - FRAME TOO SHORT
                                ; BIT 9-15 - RESERVED
        RFD.CMD=1       :RFD COMMAND
         RFD.EL=100000          ; BIT 0 - END OF LIST (EL)
         RFD.S=040000           ; BIT 1 - SUSPEND (S)
                                ; BITS 2-15 - RESERVED
        RFD.LNK=2       :RFD NEXT RFD ADDRESS
        RFD.BDP=3       :RFD BUFFER DESCRIPTOR POINTER
        RFD.DST=4       :RFD DESTINATION ADDRESS (3 WORDS)
        RFD.SRC=7       :RFD SOURCE ADDRESS (3 WORDS)
        RFD.TYP=10      :RFD TYPE FIELD

;
; LCC RECEIVE BUFFER DESCRIPTOR (RBD) FORMAT
;
         RBD.EOF=100000         ; BIT 0 - END OF FRAME (EOF)
         RBD.F=040000           ; BIT 1 - BUFFER USED (F)
                                ; BITS 2-15 BYTE COUNT
        RBD.LNK=1       :RBD NEXT RBD ADDRESS
        RBD.BUF=2       :RBD BUFFER ADDR (TWO WORDS, LOW FIRST)
        RBD.SIZ=3       :RBD CONTROL WORD
         RBD.EL=100000          ; BIT 0 - END OF LIST (EL)
                                ; BIT 1 -  RESERVED
                                ; BITS 2-15 - BUFFER SIZE (BYTES)
```

Index

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W

X

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY_____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY_____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Data General Corporation
Educational Services/TIPS
MS G214
2400 Computer Drive
Westboro, MA 01580
(617) 366-8911, Extension 1610

Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

Data General

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   Data General Corporation
   Educational Services/TIPS
   MS G214
   2400 Computer Drive
   Westboro, MA 01580
   (617) 366-8911, Extension 1610
8. We'll take care of the rest!

educational services

Data General Corporation, Westboro, MA 01580

014-000796-01