ee 200 COMPUTER REFERENCE MANUAL

technical manual

ELDORADO

ELDORADO ELECTRODATA CORPORATION

601 CHALOMAR ROAD • CONCORD, CALIF. 94520 U.S.A. • TEL. (415) 686-4200 • TWX 910-481-9476

ee 200 COMPUTER REFERENCE MANUAL

ELDORADO ELECTRODATA CORP.
601 Chalomar Road
Concord, California 94520
Telephone: 415-686-4200
TWX: 910-481-9476

TABLE OF CONTENTS

Page

ILLUSTRATIONS

1. SYSTEM DESIGN FEATURES

INTRODUCTION

This manual describes Eldorado Electrodata's ee 200 Digital Computer System. Among the items covered are: system organization, the instruction repertoire, input/output device controllers, and the memory input/output interface.

FEATURES

The ee 200 was designed to work efficiently in a real time environment. It is low in cost and is built for high reliability and ease of maintenance. The system is supported by a comprehensive software package to minimize the time required for user program development.

Some of the more specific features of the ee 200 are:

SYSTEM ARCHITECTURE

+ Universal high-speed data bus--both memory and I/O devices are attached to the same bus.

+ Modular memory capability--memory may be combinations of various speed core, integrated circuit, and read only memories to meet unique user requirements.

+ Dynamic register allocation--multiple sets of registers eliminate need for saving program counter status or operands when responding to interrupts.

+ Eight 16-bit operating registers in each register set.

+ Automatic nesting of subroutines and stack operations--both reentrant and recursive.

+ Parallel arithmetic capability, 2's complement, on either 8-bit byte or 16-bit word.

ADDRESSING

+ Up to 61,440 bytes of memory available.

+ Direct addressing of all 61,440 bytes of memory and 4,096 I/O device addresses.

+ Extensive address modification capability, including:
    - Direct
    - Indirect
    - Relative
    - Relative Indirect
    - Twelve modes of hardware index with eight index registers

INSTRUCTIONS

+ Sixty-nine basic instructions.

+ Instructions can be 8, 16, or 24 bits.

+ All instructions referring to operands operate on 8 or 16 bits.

I/O AND INTERRUPTS

+ Multiple device direct memory access (standard).

+ Maximum transfer rate at memory speed.

+ Serial ASR 33/35 interface (standard).

+ Fifteen levels of hardware priority interrupts (standard)--with unique response for each level.

+ Four sense switches.

PHYSICAL CHARACTERISTICS

+ Rack or desk mounted.

+ Environment Specifications:

- Temperature (with core memory)	0-50° C
- Temperature (with IC memory)	0-60° C
- Humidity	0-90% without condensation

+ TTL logic.

OPTIONS

+ Power failure/restart.

+ Memory and I/O parity.

+ Real-time clock.

+ Address boundary.

+ High-speed operational register files.

SOFTWARE

Software available for the ee 200 includes:

+ Standard System:

- Resident assembler
- Test operating system
- Transitional monitor
- Text editor
- ESP programming language compiler
- ESP I/O and support routines

+ Conversion/data manipulation routines.

+ Interrupt routines.

+ I/O drivers.

+ Comprehensive diagnostics.

+ Disc operating system.*

+ Communications monitor.*

PERIPHERALS AVAILABLE

+ IBM Selectric.*

+ ASR 33/35.

+ High-speed paper tape reader and punch.

+ IBM compatible magnetic tape--various transfer rates.

+ Removable media disc.

+ General-purpose interface.

+ Synchronous modem interface.

+ Asynchronous modem interface.

+ Low-cost card reader.

---

* Optional

+ Line printer, 60, 135, 600 lines per minute.

+ Digital input and output interfaces.

+ Digital cassette tape interface, 1-3 tape decks.

SPECIAL ENGINEERING SERVICES

Experienced engineering personnel are available to assist the customer in implementing special interface and software applications.

UNIVERSAL BUS

Eldorado Electrodata's ee 200 Universal Bus concept is one of the most important factors contributing to the computer's extremely versatile operational characteristics.

The basic ee 200 consists of an enclosure (ready for rack mounting) with an operator's console, three circuit boards containing the entire central processor unit, and a modular power supply. Twelve additional card slots are available for plugging in memories and input/output device controllers.

The memories and the input/output device controllers plug directly into the common Universal Bus in any sequence or combination. This means that memory and I/O devices all appear the same to the CPU--the first 61K addresses are reserved for memory; the upper 4K are reserved for I/O device addresses.

All I/O devices can access memory without going through the central processor. Direct memory access operations can be performed without costly additional hardware. In addition, peripheral devices can communicate directly with each other without processor intervention. Transfer rates are limited only by the speed of the selected memories and/or peripheral devices.

No unique input/output instructions are required in the ee 200--the memory reference instructions handle these operations.

Figure 1-1 shows the Universal Bus.

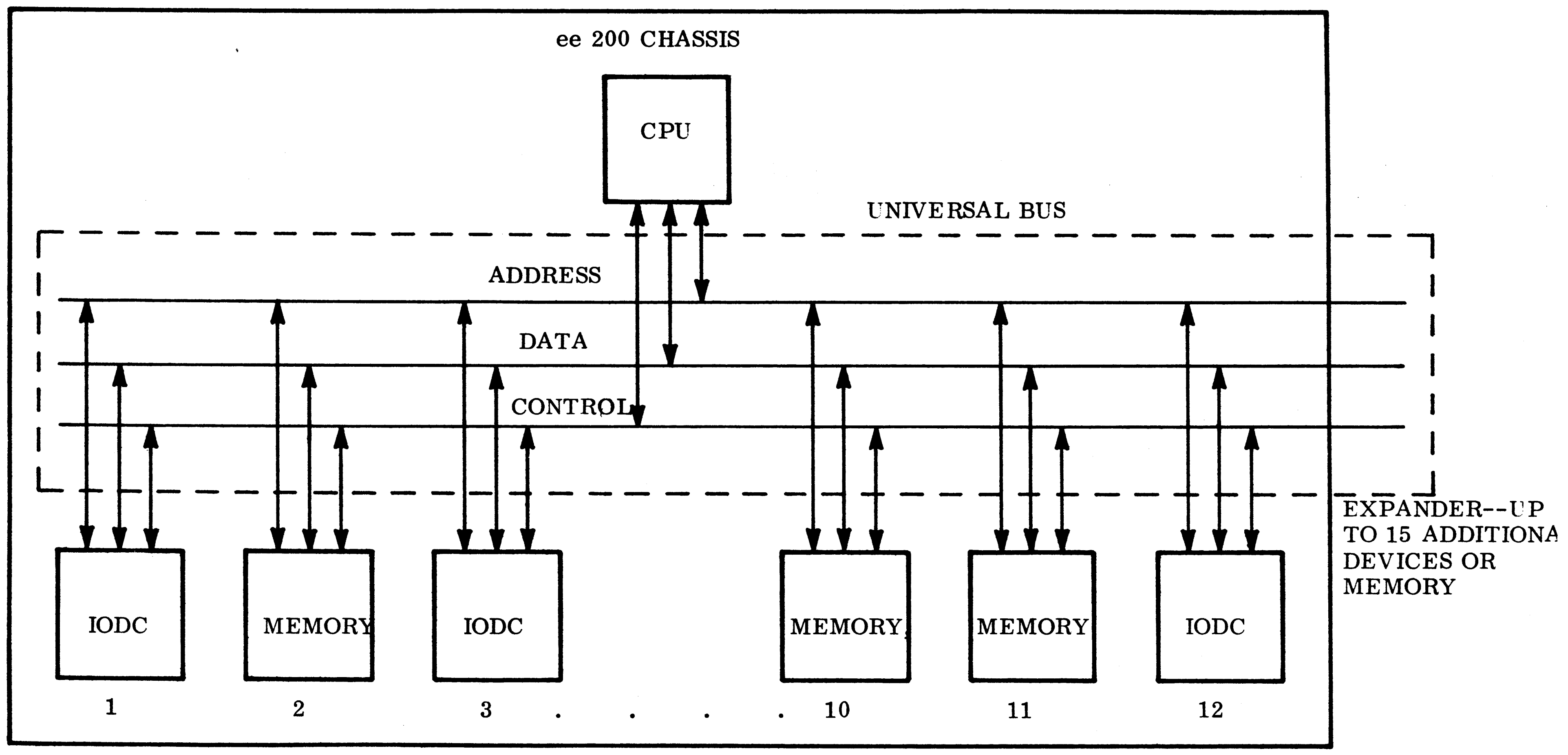


Figure 1-1. Universal Bus

ASYNCHRONOUS MEMORY INTERFACE

The asynchronous memory interface allows you to select various combinations of memory size and speed. For example, high-speed register files can be intermixed with MOS memory modules, a 1.2-microsecond core memory, and read-only memories of any capacity.

Since instruction execution time is directly related to memory cycle time, desired processor speed can be achieved by appropriate memory selection. Thus, a 16-bit add operation can range from 10.8 microseconds using core memory to 2 microseconds using a 200-nanosecond file memory.

Memories presently available include:

Magnetic Core	4K bytes
High-speed Bipolar (IC)	16, 32, 64, 128, 256, 512 bytes
Medium-Speed MOS	256, 512, 1024 bytes
Read-Only	Variable capacity

Up to 61,440 bytes of memory are available with the ee 200. All bytes are directly addressable. Figure 1-2 shows the ee 200 memory map.

INPUT/OUTPUT INTERFACE

The ee 200 Universal Bus concept makes interfacing to an input/output device a simple task. All that is required is to plug an input/output device controller into an available slot in the computer chassis and connect a cable from the chassis to the peripheral device. The connector in the chassis is already wired to the combination memory and I/O bus. The bus contains 16 address lines, 8 data lines, and several control lines.

The device controllers themselves are also simplified due to the Universal Bus. The controllers are either word or byte oriented and contain their own addressable registers. There are no special I/O instructions; memory reference instructions are used for I/O.

The controllers consist of an I/O bus interface and control section and a machine control section.

The I/O bus interface and control sections are of two types: (1) those that handle transfers under program control only; and (2) those that include automatic block transfer capability.

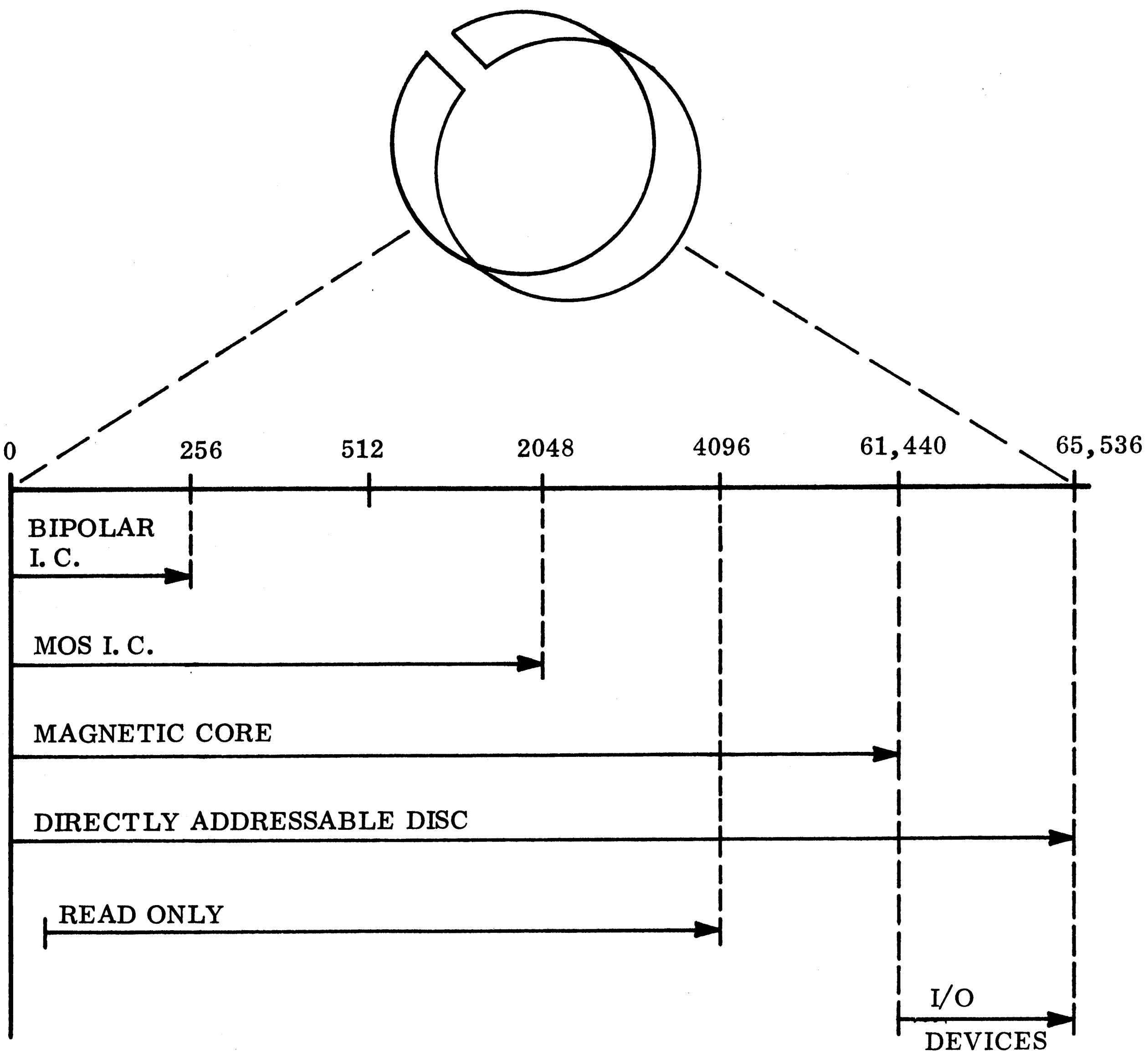


Figure 1-2. ee 200 Memory Map

Because direct memory access in the ee 200 uses the same Universal Bus as memory and I/O devices, automatic block transfer capability is a simple extension of any device controller.

The machine control circuits are different for each device to be controlled. They vary from simple registers and multiplexers for digital I/O interface boards to long sequential controllers and registers for more sophisticated machines; e.g. magnetic tape units.

The use of separate data and address lines plus the asynchronous nature of the I/O transfers eliminate complex timing and control circuits.

GLOBAL STACK POINTER

One of the eight general registers (the S register) assigned to the background processing level can be used as an index or working register. However, its primary function is to maintain a stack during subroutine exits and entries which provides subroutine reentrancy in interrupt environments. The S register value for level 0 is global. That is, S is common to all 16 general register sets.

A stack is a time-ordered set of operands in memory. The S register is the pointer used to define the last element of the stack. The stack is ordered backwards in memory. That is, the first element in is the last element out. Thus, the S register is initially set to the value that is the starting point of the stack and, as operands are pushed onto the stack, S is decremented. When operands are pulled from the stack, S is incremented. When a "Jump to Subroutine" instruction is encountered, the contents of the X register are pushed onto the stack and the program counter value is loaded into the X register. On return from subroutine, the contents of X are loaded into the program counter and X is loaded from the stack.

The stack can also be used for temporary storage of operands from the A, B, or X registers. Figure 1-3 shows the stack operation.

INTERRUPT HANDLING

The ee 200 was designed for real-time systems application. Handling a real-time environment involves rapid context switching, that is, moving from one process to another with a minimum of overhead in time and instructions. The usual approach is to allow a "Jump and Save" instruction to save the current program counter value. Switching of flags (fault, zero, etc.) and storing of current operands requires significant hardware or several software steps which must be noninterruptible.

1. PROGRAM SETS S TO INITIAL VALUE.
2. S IS DECREMENTED BY ONE--FIRST BYTE PUSHED ONTO STACK AS RESULT OF "STAB (S-)" INSTRUCTION.
3. S IS DECREMENTED BY TWO--FIRST WORD (X) PUSHED ONTO STACK AS RESULT OF "JSR MUL" INSTRUCTION.
4. S IS DECREMENTED BY ONE--SECOND BYTE PUSHED ONTO STACK AS RESULT OF "STAB (S-) INSTRUCTION, E.G., PARTIAL PRODUCT DURING MULTIPLY SUBROUTINE.
5. INTERRUPT OCCURRED, ENCOUNTERED "JSR MUL" SUBROUTINE WITHIN INTERRUPT. S IS DECREMENTED BY TWO--SECOND WORD (X) PUSHED ONTO STACK.
6. S IS DECREMENTED BY ONE--THIRD BYTE PUSHED ONTO STACK AS RESULT OF "STAB (S-)" INSTRUCTION. AGAIN, A PARTIAL PRODUCT DURING MULTIPLY SUBROUTINE.
7. THIRD BYTE IS PULLED FROM STACK AS RESULT OF "LDAB (S+)" INSTRUCTION--S IS INCREMENTED BY ONE.
8. "RSR" INSTRUCTION CAUSES RETURN FROM SUBROUTINE WITHIN INTERRUPT. SECOND WORD IS PULLED FROM STACK--S IS INCREMENTED BY TWO.
9. RETURNED FROM INTERRUPT BACK TO MULTIPLY SUBROUTINE. SECOND BYTE IS PULLED FROM STACK AS RESULT OF "LDAB (S+)" INSTRUCTION--S IS INCREMENTED BY ONE.
10. "RSR" INSTRUCTION CAUSES RETURN FROM SUBROUTINE TO BACKGROUND PROGRAM. FIRST WORD IS PULLED FROM STACK--S IS INCREMENTED BY TWO.
11. FIRST BYTE IS PULLED FROM STACK AS RESULT OF "LDAB (S+)" INSTRUCTION--S IS INCREMENTED BY ONE AND RETURNED TO ITS INITIAL VALUE.

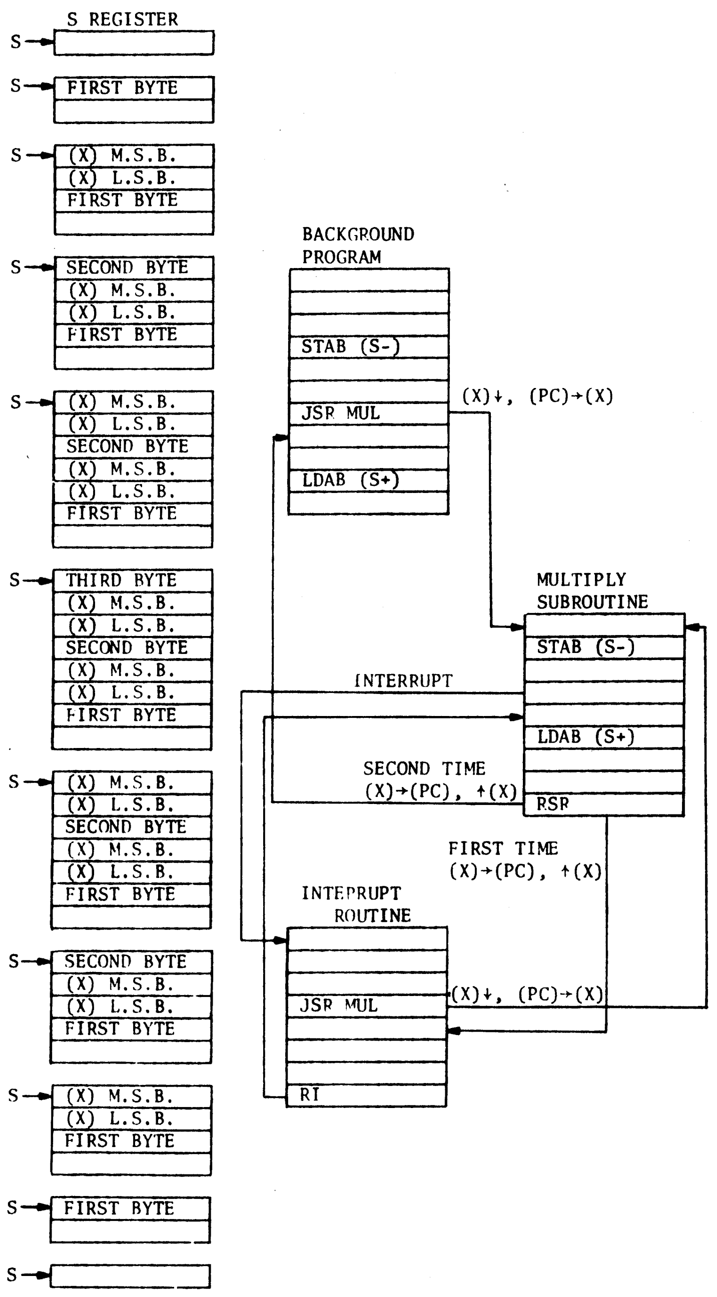


Figure 1-3. ee 200 Stack Operation

In the ee 200 Computer, this entire process is eliminated. Context switching is accomplished entirely automatically with no need to save and restore register values and flags.

The operational registers, accumulators, index registers, etc., are addressable in memory locations. Normal background processes are conducted with the operational registers residing in memory locations 0 through F (the first sixteen bytes). The fifteen sets of priority interrupt registers are allocated to successive locations in memory. The fifteen interrupt levels are designated IL01 through IL0F, with IL01 having the lowest priority and IL0F having the highest.

During competitive situations, a higher priority level always gains access ahead of a lower priority. If entry has already been made to a lower priority handling routine, the higher level can immediately draw the processor out of the lower level. When the higher level is complete, a linkage is available to allow the processor to thread back to the lower level program. A similar capability can often be claimed for other machines but it is typically effected with significant software overhead. This software overhead manifests itself in real time during which the machine is nonresponsive to higher levels. In the ee 200 Computer this time is never more than one instruction time.

UNLIMITED FUNCTIONAL EXPANSION

No expensive software or hardware modifications to the CPU are required to perform a special function. All that is required is to plug a "black box" into the Universal Bus and assign an address to it. When the "black box" is addressed and given an execute command, the CPU halts, waits for the function to be performed, and then resumes processing. All devices on the Universal Bus have direct access to the computer's high-speed general registers, thereby allowing for a variety of applications.

Some examples of what the "black box" might be designated to do are:

- Logical functions
- Multiply
- Divide
- Sort
- Table look-up
- Table search
- Hardware algorithms (such as sample 128 Teletype lines)

It is readily apparent that this feature effectively gives the user an unlimited "instruction list," thereby allowing you to inexpensively customize your system.

HARDWARE BOOTSTRAP LOADER

A hardware bootstrap loader is furnished as a standard feature in the ee 200 Computer.

POWERFUL INSTRUCTION SET--EXTENSIVE ADDRESS MODIFICATION

The ee 200 has a powerful instruction set with the most extensive address modification capability of any mini computer. The instruction list includes 8, 16, and some 24-bit instructions. There are 69 basic commands and hundreds of permutations.

The address modes include:

1. Direct addressing of any I/O device or up to 61K of memory.
2. Fully extended indirect addressing.
3. Relative addressing.
4. Relative indirect.
5. One and two-byte literals.
6. Indexed addressing.

Any of the eight general-purpose registers can be used as index registers.

The twelve indexing modes are:

1. Index.
2. Index with auto-increment.
3. Index with auto-decrement.
4. Index indirect.
5. Index indirect with auto-increment.
6. Index indirect with auto-decrement.
7. Index with displacement.
8. Index with displacement and auto-increment.
9. Index with displacement and auto-decrement.

10. Index with displacement indirect.

11. Index with displacement indirect and auto-increment.

12. Index with displacement indirect and auto-decrement.

The instructions can be categorized as follows:

Control (15)

Conditional Branches (16)

Single Register (16)

Double Register (14)

Memory Reference (8)

Both single register and double register instructions can operate on byte or word operands.

2. SYSTEM ORGANIZATION

INTRODUCTION

Eldorado Electrodata's ee 200 is a general-purpose digital computer. Up to 61,440 bytes of memory can be directly addressed. Communication between the central processor, memory and I/O devices is accomplished on a single common high-speed Universal Bus. The Universal Bus consists of address, data, and control lines. Input and output share the same data lines. The central processor uses the bus to read and write from/to memory and I/O devices. The maximum transfer rate is limited only by memory cycle time.

GENERAL REGISTERS

Eight 16-bit general registers occupying the first 16 bytes of addressable memory are used for normal background processing. In addition, there are eight 16-bit general registers for each of the 15 interrupt levels. These registers reside in the next 240 bytes of addressable memory.

Each register can be used in the word mode as a full 16-bit register or in the byte mode as two 8-bit registers. Figure 2-1 shows the general registers for level zero. The even-numbered register address refers to the most-significant byte; the odd-numbered address refers to the least-significant byte. When an interrupt occurs, the contents of the value, minus, fault, and link indicators are stored in the four high-order bits in the most-significant byte of the C register; the last interrupt level is stored in the four high-order bits in the least-significant byte of the C register.

In the byte mode either the left or the right byte may be specified. When using the word mode, each register is referred to by the register number of its left-most (most-significant) byte. For example, the A register is specified by "0". If the register number of the right-most byte is specified, the right-most byte is used twice to form a 16-bit operand.

REGISTER ADDRESS	REGISTER NAME	
	(M.S.B.)	(L.S.B.)
0-1	A	A
2-3	B	B
4-5	X	X
6-7	Y	Y
8-9	Z	Z
A-B	S	S
C-D	C	C
E-F	P	P

Figure 2-1. General Registers, Level 0

Certain byte instructions imply a specific register. In these cases, it is the right-most byte that is implied.

Although all general registers may be used as index registers, accumulators, counters, etc., certain of the registers have functions or implied usages as follows:

Register	Usage
A	Primary accumulator
B	Secondary accumulator
X	Primary index
Y	Secondary index or working register
Z	Secondary index or working register
S	Stack pointer
C	Context register
P	Program counter base

A AND B REGISTERS

The contents of the A and B registers may be loaded or stored (byte or word) from/to memory using the full address modification power of the ee 200. Also, these registers are implied in most of the single-byte instructions.

X REGISTER

The X register can be loaded or stored from/to memory using less extensive address modification power than that for A and B. All X register operations are 16-bits (word mode).

Y AND Z REGISTERS

The Y and Z registers can be used as index registers or as working storage registers with A, B, and X.

S REGISTER

The S register is a general register capable of use as an index or working register. In addition, the S register for level 0 is used to maintain a stack during subroutine exits and entries which provides subroutine reentrancy in interrupt environments. The S register value for level 0 is global. That is, S is common to all general register sets.

C REGISTER

The C register contains the settings for the status condition indicators (fault, link, minus, and value) and the interrupt level which preceded the current level.

P REGISTER

The P register contains the initial program counter value to be used when starting up and on return from an interrupt subroutine.

All of the general registers are fully usable in single and double register operations as well as for general index registers in the memory reference instruction group.

MEMORY MODULES

The flexibility of the ee 200 provides the user with a choice of memory types. Different types of memories can be mixed in the same machine.

Each memory unit requires a single printed circuit card connection to the memory and I/O bus (Universal Bus). Any memory module can be inserted into the bus at any bus connector.

COMPUTER OPTIONS

There are several components to a computer system. These fall generally into the categories of basic computer, basic computer options, memories, and I/O device controllers.

The basic ee 200 Computer consists of:

+ The Central Processing Unit (with three printed-circuit boards).
+ Chassis (with printed-circuit mother board interconnect).
+ Power Supply.
+ Basic Control Panel.

Options available for the basic computer are:

+ Central Processor Option Board.
    - Power Fail/Automatic Restart
    - Memory Parity
    - Real-Time Clock
    - Address Boundary
+ Augmented Control Panel.
+ Memory Options.
+ Input/Output Device Controller Options.

CENTRAL PROCESSOR OPTION BOARD

The CPU option board can contain the power fail/automatic restart option, the real-time clock option, the memory parity option, and the address boundary option. When an interrupt is generated by any one of these options, the processor switches to interrupt level 15 which contains the address of an interrupt subroutine. The subroutine checks the status of the option board to determine which option initiated the interrupt. When this is determined, a jump to the appropriate subroutine is executed.

The power fail/automatic restart option provides an interrupt when power is initially turned on and when a loss of primary power is detected. This feature protects an operating program by storing all volatile registers in memory when the interrupt occurs. When power is restored, a second interrupt is generated and all registers are restored to their previous condition. Normal processing is then resumed.

The real-time clock option provides a method of accurately measuring time intervals. The clock frequencies of 10 Hz, 100 Hz, 1 KHz, and 10 KHz are selectable under program control. In addition, the 1 KHz counter can be programmed to generate other frequencies. This is accomplished by loading the accumulator with a number specifying the number of 1-KHz counts to be generated between interrupts. This number is complemented as it is loaded into a count register in the real-time clock logic. When the count register overflows, an interrupt occurs.

The parity option contains a parity generator and a parity check circuit for checking each word written into or read from memory and selected input/output devices. Even parity is used. If a byte containing an odd number of "ones" is detected, an interrupt is generated.

The address boundary option provides an indication to the computer if a nonexistent memory location or input/output device address is given. When this occurs, an interrupt is generated and a pseudo data ready signal (DRDY) is given to the processor. The DRDY signal is required because of the closed-loop communication used by the processor. Each time the computer issues a command, it pauses until it receives a response.

The basic computer contains a generator that provides a pseudo DRDY signal after a 4.55-millisecond pause. This generator is disabled when the address boundary option is included.

MEMORY OPTIONS

Memory options include Read-Write Memories (RWM) as well as Read-Only Memories (ROM) of various cycle times (see table 2-1).

I/O DEVICE OPTIONS

Device controllers are available for the following input/output options:

+ IBM Selectric.

+ ASR 33/35.

+ High-speed paper tape reader and punch.

+ IBM compatible magnetic tape--various transfer rates.

+ Removable media disc.

+ General-purpose interface.

+ Synchronous modem interface.

+ Asynchronous modem interface.

+ Low-cost card reader.

+ Line printer, 60, 135, 600 lines per minute.

+ Digital input and output interfaces.

+ Digital cassette tape interface, 1-3 tape decks.

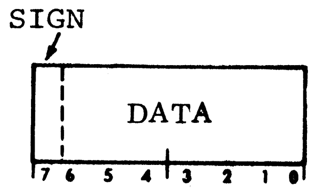
Table 2-1. Memory Modules Available

Size	Type	Cycle Time (Microseconds)
4096 bytes	Magnetic Core RWM	1.2
1024 bytes	Magnetic Core RWM	1.2
256 bytes	MOS Integrated Circuit RWM	1.2
512 bytes	MOS Integrated Circuit RWM	1.2
1024 bytes	MOS Integrated Circuit RWM	1.2
16 bytes	Bipolar Integrated Circuit RWM	0.2
32 bytes	Bipolar Integrated Circuit RWM	0.2
64 bytes	Bipolar Integrated Circuit RWM	0.2
128 bytes	Bipolar Integrated Circuit RWM	0.2
512 bytes	Custom ROM	1.2
1024 bytes	Custom ROM	1.2

DATA FORMATS

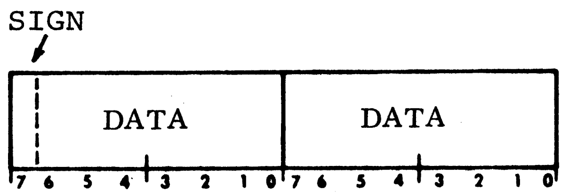
Data is handled as one or two bytes, depending upon the instruction. Any operation code that references an operand can do so with either 8- or 16-bit precision.

8-Bit Data Format:



Binary Two's Complement; Range: $-2^7 \leq \eta < 2^7$

16-Bit Data Format:



Binary Two's Complement; Range: $-2^{15} \leq \eta < 2^{15}$

INSTRUCTION FORMATS

The instruction formats vary in length from one to three bytes. In all cases, the most-significant byte consists of an operation code which defines the class of instruction and the operation to be performed. Additional bytes specify such things as jump location in the case of branch instructions, source and destination registers, incrementation and decrementation. For memory reference instructions the additional bytes contain address data, register data, and address modification information. Most instructions can be used in either the word mode or the byte mode. In these cases, the operation code for the word mode is shown first, followed by the code for the byte mode in parenthesis. When using the byte mode, the least-significant half of the designated register is assumed.

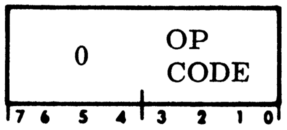
The instructions can be categorized into the following groups:

- Control
- Branch
- Single Register
- Double Register
- Memory Reference

The formats for these groups are given in the following paragraphs.

CONTROL INSTRUCTIONS

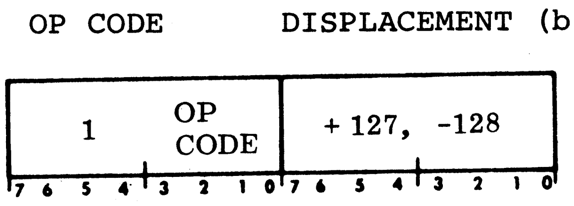
The control group instructions are single-byte instructions which provide specific control functions. Control instructions have the following format:



The 0 in the most-significant half of the byte indicates that the instruction is a control instruction. The least-significant half of the byte defines the instruction.

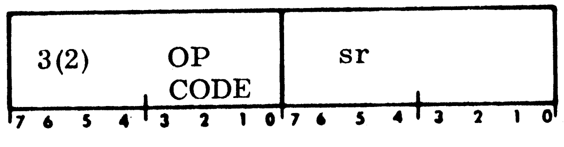
BRANCH INSTRUCTIONS

The branch instructions have a two-byte format. The operation code byte defines the condition being tested. The displacement byte, b, contains an eight-bit signed value which specifies a jump location relative to PC. The branch has a range of +127 bytes and -128 bytes with reference to the current value in PC. At the time of execution, the PC contains the address of the next instruction following the branch instruction. The displacement value is summed with the current contents of the PC register if the test condition is met.



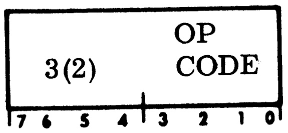
SINGLE REGISTER OPERATIONS

There are two types of single-register operations, explicit and implicit. Explicit register instructions are two-byte instructions with the source register address contained in the second byte. (See figure 2-1 for register designations.) The form of an explicit single-register instruction is:



The four low-order bits of the second byte are used only for the Increment Register and Decrement Register instructions and are otherwise ignored.

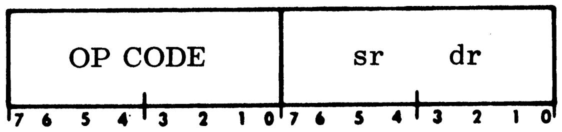
Implicit register instructions are single-byte instructions with the source register implicit in the operation code. The form of an implicit single-register instruction is:



The least-significant byte (8 bits) of a 16-bit word is implied when using the byte mode.

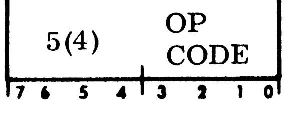
DOUBLE REGISTER OPERATIONS

There are two types of double-register operations, explicit and implicit. Explicit register instructions are two-byte instructions with the source register and destination register addresses contained in the second byte. The form of an explicit double-register instruction is:



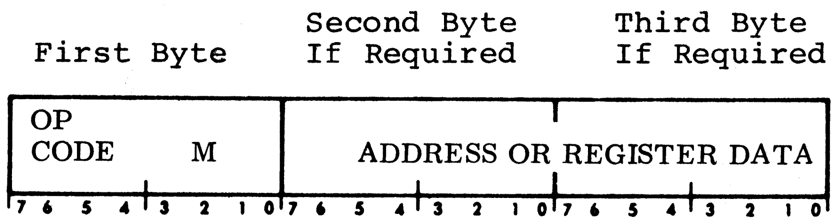
The operation code may specify either a byte or full word operation. In the byte mode, the least-significant byte is assumed. The source register and destination register may be the same. See Figure 2-1 for register designations.

Implicit register instructions are single-byte instructions with the source and destination registers implicit in the operation code. The form of an implicit double-register instruction is:



MEMORY REFERENCE INSTRUCTIONS

Memory reference instructions are those which access an operand from addressable memory. They can operate on either byte operands or word operands. When operating in the byte mode, the least-significant half of the register is used. The instructions can be 8, 16, or 24 bits long, depending upon the address modification used. The form of the memory reference instruction is:

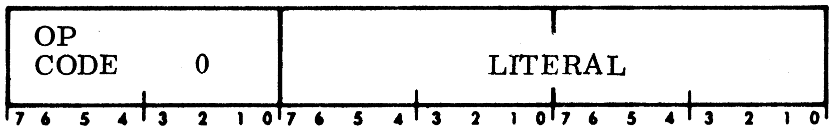


ADDRESS MODIFICATION

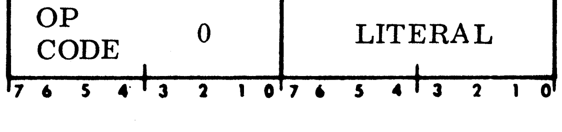
Address modification capability varies with the type of instruction and is a function of the M field. The codes used in the M field and their effects on the address are discussed in the following paragraphs.

Basic Address Modification

M = 0 Literal

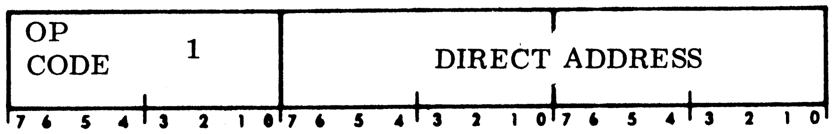


With a word instruction, two bytes following the operation code either contain the literal word or receive the literal word from A, B, or X.



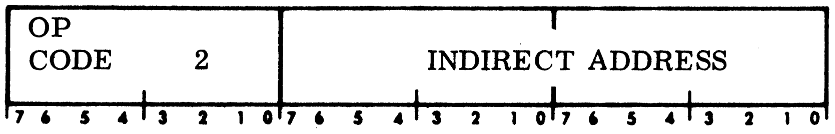
With a byte instruction, one byte following the operation code either contains the literal byte or receives the literal byte.

M = 1 Direct



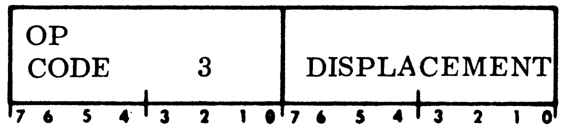
With a word or byte memory reference instruction, two bytes following the operation code contain a full 16-bit address which is used to directly address the operand(s). The 16-bit address allows direct addressing to $65,536_{10}$ addresses. This gives the facility for addressing memory (up to $61,439_{10}$), and I/O device controllers ($61,440_{10}$).

M = 2 Indirect



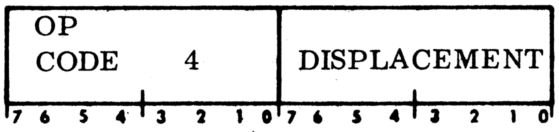
With a word or byte memory reference instruction, 2 bytes following the operation code contain a full 16-bit indirect address. The operand(s) are located by using the contents of the 2-byte memory location pointed to by the indirect address value as an absolute address.

M = 3 Relative to Current Location Counter



With a word or byte memory reference instruction, the address of the operand(s) is constructed by adding the contents of the byte following the operation code (the displacement) to the contents of the program counter. The displacement is in two's complement notation allowing access to operands +127 bytes ahead or -128 bytes behind the value in the program counter. At the time of execution, PC contains the address of the next instruction following.

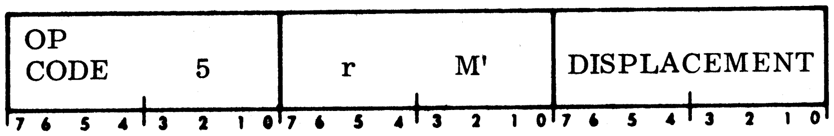
M = 4 Relative to Current Location Counter, Indirect



With a word or byte memory reference instruction, the address of the operand(s) is constructed by adding the contents of the byte following the operation code to the contents of the program counter as with M = 3. At this point, however, the 16-bit result is interpreted as pointing to another 2-byte address which is used as the actual operand address.

M = 5 Indexed Addressing

When a word or byte instruction has an M code of 5, the byte or bytes following are interpreted as register address modes. The displacement byte is only included in those modes which require it. The general form for indexed addressing is:



r refers to the base register as defined in figure 2-1.

M' is the new modification code and is defined as follows:

0	Use register directly as address
1	Use register as address and increment after use
2	Decrement register and use as address
3	Not used
4	Use register as indirect address
5	Use register as indirect address and increment after use
6	Decrement register and use as indirect address
7	Not used

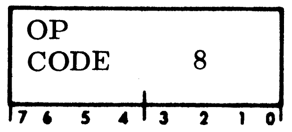
8	Add displacement byte to register and use as direct address
9	Add displacement byte and use as direct address, then increment register
A	Decrement register, and displacement byte, and use as direct address
B	Not used
C	Add displacement to register, and use as direct address
D	Add displacement byte, and use as indirect address, then increment register after use
E	Decrement register, add displacement byte, and use as indirect address
F	Not used

Incrementation and decrementation will be by one or two, depending upon whether the instruction is a byte or word instruction.

Implicit Indexing (One Byte)

When using implicit indexing, the M code can be 8 through F. With a word or byte memory reference instruction, the operand(s) address is the 16-bit contents of the designated register. The format shown is for M = 8.

M = 8 Register Addressing - A Register



The M field codes and operand addresses for the instructions are:

M Field	Operand(s) Address
8	A Register
9	B Register
A	X Register
B	Y Register
C	Z Register
D	S Register
E	C Register
F	P Register

EFFECTIVE ADDRESS COMPUTATION

Table 2-2 shows the method used to arrive at the effective address. The symbol Δ as used in the table is equal to one or two, depending upon whether the instruction is byte or word mode.

Table 2-2. Effective Address Computation

M	Effective Address	Mode
0	Second and/or third byte of instruction	Literal
1	(Second and third byte of instruction)	Direct
2	((Second and third bytes of instruction))	Indirect
3	(PC)+b	Relative
4	((PC)+b)	Relative Indirect
5 M'		
0	(r)	Index
1	(r)	Index; Index Returned Incremented by Δ

5	M'	Effective Address	Mode
	2	(r)- Δ	Decrement and Index
	4	((r))	Index Indirect
	5	((r))	Index Indirect; Index Re-turned Incremented by Δ
	6	((r)-Δ)	Decrement and Index In-direct
	8	(r)+b	Index with Displacement
	9	(r)+b	Index with Displacement; Index Returned Incremented by Δ
	A	(r)-Δ+b	Decrement and Index with Displacement
	C	((r)+b)	Index with Displacement In-direct
	D	((r)+b)	Index with Displacement In-direct; Index Returned Incremented by Δ
	E	((r)-Δ+b)	Decrement and Index with Displacement Indirect
8		(A)	Indexed by A Register
9		(B)	Indexed by B Register
A		(X)	Indexed by X Register
B		(Y)	Indexed by Y Register
C		(Z)	Indexed by Z Register
D		(S)	Indexed by S Re[illegible]
E		(C)	Indexed by C [illegible]
F		(P)	[illegible] by [illegible]

3. INSTRUCTION SET

INTRODUCTION

The basic instruction set for the ee 200 Computer System consists of 69 instructions. Descriptions of these instructions, grouped according to function, are given in this section. Refer to Appendix A for a list of instructions.

INSTRUCTION EXECUTION TIME

Instruction execution time is a function of the type of memory selected. Some typical times are given in table 3-1. All times are in microseconds.

Table 3-1. Typical Instruction Execution Times

Instruction Type	Memory Type		
	High-Speed Files Only*	Core Only	Core With Files**
16-Bit Load	4.6	9.4	7.8
16-Bit Add	4.4	10.8	6.0
16-Bit Single Register Increment, Decrement, Complement, Clear	4.0	8.8	5.6

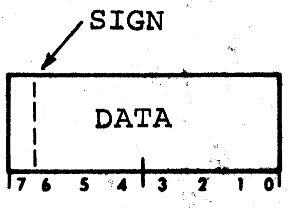
* High-speed files used for operational registers as well as storage; i.e., greater than 16 bytes.

** Files used for operational registers only.

DATA FORMATS

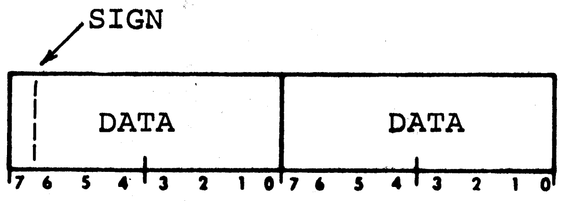
Data is handled as one or two bytes, depending upon the instruction. Any operation code that references an operand can do so with either 8- or 16-bit precision.

8-Bit Data Format



Binary Two's Complement; Range: $-2^{15} \leq n < 2^{7}$

16-Bit Data Format



Binary Two's Complement; Range: $-2^{15} \leq n < 2^{15}$

INSTRUCTION DESCRIPTIONS

All instructions are assigned a two- or a three-character alphanumeric mnemonic. The first character of the mnemonic must be an alpha character. For example, the Load A instruction mnemonic is LDA.

Byte operations are indicated by adding the letter B to the instruction mnemonic. In the absence of B, word operation is assumed. For example:

LDA	Load A - word mode (16-bits)
LDAB	Load A - byte mode (8-bits)

In the instruction descriptions, the letter B is added to the mnemonic in parenthesis to indicate that the instruction can be used in either the byte or the word mode. For example, the Load A instruction mnemonic is shown as LDA(B). The operation code for the byte mode is shown in parenthesis following the code for the word mode.

CONDITION FLAGS

There are four condition flags used in conjunction with the computer operations. They are: Fault (F), Link (L), Minus (M), and Value (V). These flags may be tested (conditional branch instructions) to determine the result of an operation upon data. The fault and link may also be used as programming flags or indicators.

All condition flags are set or reset by the contents of the context (C) register when an interrupt occurs. The condition status of the interrupted level is saved and then restored when the program returns to that level. Also, specific control instructions are provided to set and reset the fault and link condition indicators.

The fault indicator is reset by the clear instruction and by the reset fault instruction. The set fault instruction sets the indicator. For increment, decrement, and arithmetic operations, it is set if the sign of the result (destination register) changes. For left shift and rotate instructions, the setting of fault is the exclusive OR function of the link and minus flags after the shift occurs. Thus, fault is set if the sign bit changes. This feature is used primarily for normalization.

The link indicator is reset by the clear instructions and by the reset link instruction. It is set by the set link instruction and complemented by the complement link instruction. During shift or rotate instructions, the link indicator is set if a one is shifted into the link, otherwise, it is reset. For add operations, the indicator is first reset and then set if the carry out of the most-significant bit is a one. For subtract instructions, it is first reset and then set if there is no carry out.

The minus indicator is reset by single register, double register, and memory reference instructions and is then set if there is a one in the most-significant bit of the source register, destination register, or operand, respectively.

The value indicator is also reset by single register, double register, and memory reference instructions, and is then set if the result equals zero.

The settings of the condition flags as the result of individual instructions are shown in Appendix A.

NOTATION

Symbolic notation used in the instruction descriptions is as follows:

General Registers	Descriptions
A	General register A
B	General register B
X	General register X
Y	General register Y
Z	General register Z
S	Stack register
C	Context register
P	General register P

CPU Registers	Description
F	Fault condition indicator
I	Interrupt flip-flop
L	Link condition indicator
LV	Interrupt level select register
M	Minus condition indicator
MA	Memory address register
PC	Program counter
R	Run flip-flop
V	Value condition indicator

Descriptive Notation	Description
()	Contents of specified register, field, etc.
$\rightarrow$	Replaces
+	Addition

Descriptive Notation	Description
-	Subtraction or negation
Λ	Logical AND
V	Logical OR
$\not{V}$	Logical exclusive OR
$\bar{X}$	Bar over symbol indicates one's complement
b	Byte
EA	Effective address
r	Any general register
sr/dr	Source register/Destination register
r_{1-3}	Subscript indicates bit(s) affected
x	Used in subscripts to indicate 8 or 16 bits
$\uparrow$	"Pop" from the stack: ((s)) (r), (s)+ (s)
$\downarrow$	"Push" into the stack: (s)- (s), (r) ((s)) Δ=1 or 2 depending on instruction, byte or word.

Each instruction description includes the following:

Instruction mnemonic

Name of instruction

Pictorial representation of instruction

Hexadecimal operation code

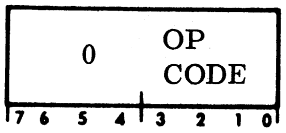
Instruction description

Affected registers, condition indicators, etc.

Symbolic operation statement

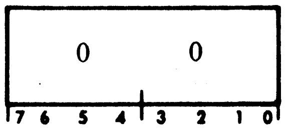
CONTROL INSTRUCTIONS

The control group instructions are single-byte instructions which provide specific control functions. Control instructions have the following format:



The 0 in the most-significant half of the byte indicates that the instruction is a control instruction. The least-significant half of the byte defines the instruction.

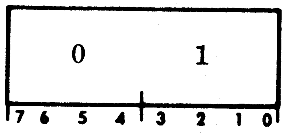
HLT - Wait for Interrupt (Halt)



The processor is halted. The content of the P Register is the address of the half instruction plus one. The RUN state is entered when the CPU receives an interrupt or when the RUN switch on the control panel is activated.

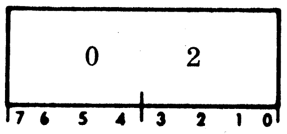
(PC)→(P), O→R   Affected: P, R

NOP - No Operation



No operations are performed by this instruction.

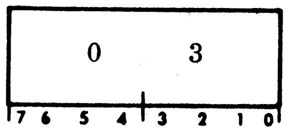
SF - Set Fault



The fault condition flag is set to one.

1→(F)   Affected: F

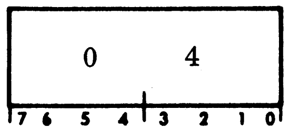
RF - Reset Fault



The fault condition flag is set to zero.

0→(F)   Affected: F

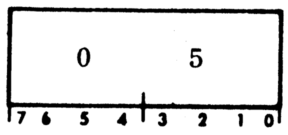
EI - Enable Interrupt System



The interrupt system is enabled, allowing the processor to recognize all external interrupts.

1→(I)   Affected: I

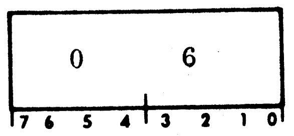
DI - Disable Interrupt System



The interrupt system is disabled, preventing the processor from recognizing any external interrupts.

0→(I)   Affected: I

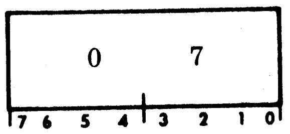
SL - Set Link



The link condition flag is set to one if its state is zero, and is unaffected if its state is one.

$1 \rightarrow (L)$ Affected: L

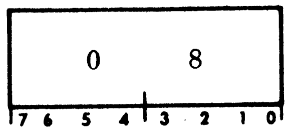
RL - Reset Link



The link condition flag is set to zero if its state is one, and is unaffected if its state is zero.

$0 \rightarrow (L)$ Affected: L

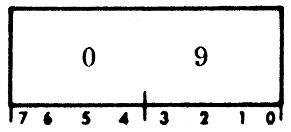
CL - Complement Link



The state of the link condition flag is complemented.

$(\bar{L}) \rightarrow (L)$ Affected: L

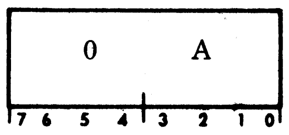
RSR - Return from Subroutine



Used to exit from a subroutine. The contents of X is placed in the program counter and X is loaded from the top of the stack.

$(X) \rightarrow (PC)$, $\uparrow (X)$ Affected: X, PC, S

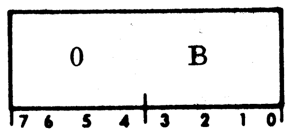
RI - Return from Interrupt



Used to exit from an interrupt service routine. The current priority level is deactivated after saving the program counter and status in the P and C registers for that level. The previously active priority level is reactivated and executed.

$(PC) \rightarrow (P)$, Status C_{12-15}, C_{4-7} LV Affected: P, C, LV, F, L, M, V

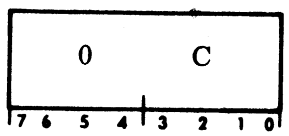
RIM - Return from Interrupt Modified



Used to exit from an interrupt service routine. The current priority level is deactivated after saving the status for that level in the C register. The program counter is not saved. The previously active level is reactivated and executed.

Status $\rightarrow C_{12-15}$, C_{4-7} LV Affected: C, LV, F, L, M, V

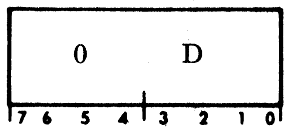
ELO - Enable Link to Teletype



This instruction is used to output to the serial Teletype interface. It transfers the status of the L condition indicator to the Teletype. If L equals one, the output Teletype line is set to the "MARK" condition. If L equals zero, the line is set to the "SPACE" condition. Upon completion of the transmission, L should be set to one and a final ELO issued to prevent the Teletype from chattering. See section four for a description of the serial Teletype interface.

$(L) \rightarrow$ Serial TTY Interface Affected: None

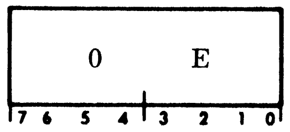
PCX - Move PC to X



The current value in PC is moved to the X register. The value in PC at the time of the move is the address of the next instruction following PCX. The next instruction in sequence is then executed. This instruction is used to access the current value of PC in "relocatable" programs.

(PC)$\rightarrow$(X) Affected: X

DLY - Delay

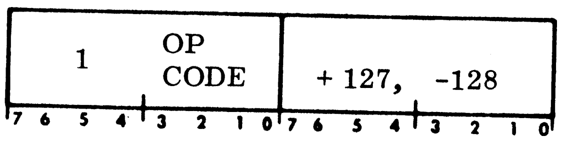


The processor is halted for 4.55 milliseconds. No processor activity occurs during the halt. This instruction is used to mark time during output to the Teletype.

Delay 4.55 milliseconds Affected: None

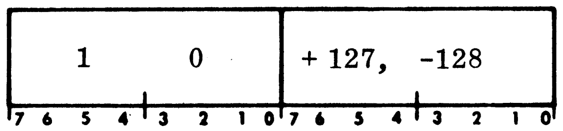
BRANCH INSTRUCTIONS

The branch instructions have a two-byte format. The operation code byte defines the condition being tested. The displacement byte, b, contains an eight-bit signed value which specifies a jump location relative to PC. The branch has a range of +127 bytes and -128 bytes with reference to the current value in PC. At the time of execution, the PC contains the address of the next instruction following the branch instruction. The displacement value is summed with the current contents of the PC register if the test condition is met.



The one in the most-significant half of the first byte indicates that the instruction is a branch instruction. The least-significant half of the first byte defines the instruction. Conditional branches have no effect on condition codes or general registers.

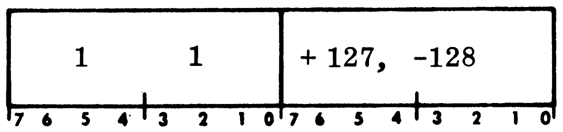
BL - Branch if Link Set



The content of the link condition flag is tested and if it contains a one, a branch is made. If the link condition flag contains a zero, no branch is made and the next sequential instruction is initiated.

If L=1, (PC)+b→(PC)

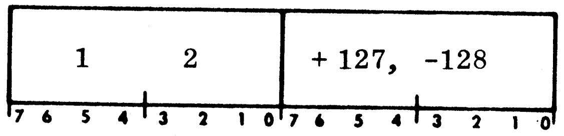
BNL - Branch if Link Not Set



The content of the link condition flag is tested and if it contains a zero a branch is made. If the link condition flag contains a one, no branch is made and the next sequential instruction is initiated.

If L=0, (PC)+b→(PC)

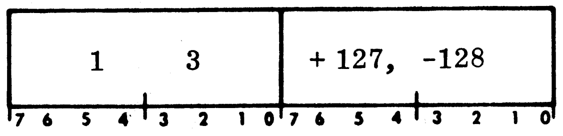
BF - Branch if Fault Set



The content of the fault condition flag is tested and if it contains a one, a branch is made. If the fault condition flag contains a zero, no branch is made and the next sequential instruction is initiated.

If F=1, (PC)+b→(PC)

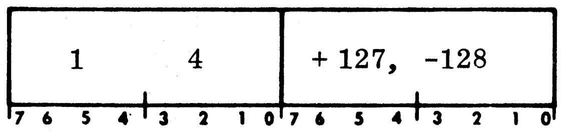
BNF - Branch if Fault Not Set



The content of the fault condition flag is tested and if it contains a zero, a branch is made. If the fault condition flag contains a one, no branch is made and the next sequential instruction is initiated.

If F=0, (PC)+b→(PC)

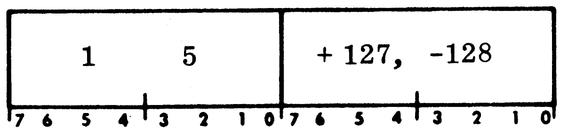
BZ - Branch if Equal to Zero



The content of the value condition flag is tested and if it contains a one, a branch is made. If the value condition flag contains a zero, no branch is made and the next sequential instruction is initiated.

If V=1, (PC)+b→(PC)

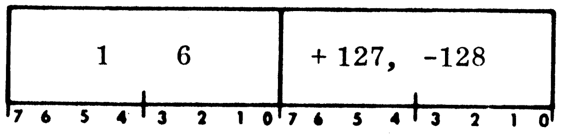
BNZ - Branch if Not Equal to Zero



The content of the value condition flag is tested and if it contains a zero, a branch is made. If the value condition flag contains a one, no branch is made and the next sequential instruction is initiated.

If V=0, (PC)+b→(PC)

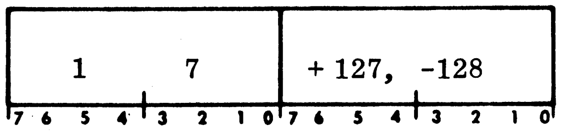
BM - Branch if Minus Set



The content of the minus condition flag is tested and if it contains a one, a branch is made. If the minus condition flag contains a zero, no branch is made and the next sequential instruction is initiated.

If M=1, (PC)+b→(PC)

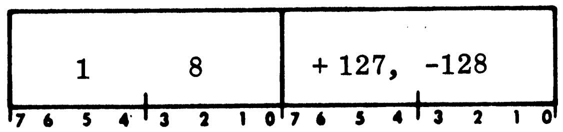
BP - Branch on Plus



The content of the minus condition flag is tested and if it contains a zero, a branch is made. If the minus condition flag contains a one, no branch is made and the next sequential instruction is initiated.

If M=0, (PC)+b→(PC)

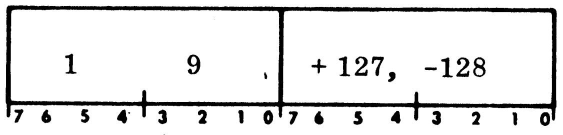
BGZ - Branch if Greater than Zero



The contents of the minus flag and value flag are both tested and if they both contain zeros, a branch is made. If either the minus or value flag contains a one, no branch is made and the next sequential instruction is initiated.

If M=V=0, (PC)+b→(PC)

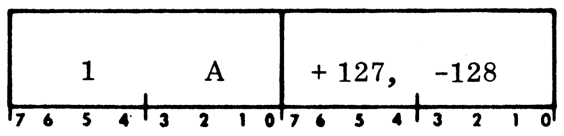
BLE - Branch if Less Than or Equal to Zero



The contents of the minus flag and the value flag are both tested and if either flag contains a one, a branch is made. If both flags contain a zero, no branch is made and the next sequential instruction is initiated.

If M or V=1, (PC)+b→(PC)

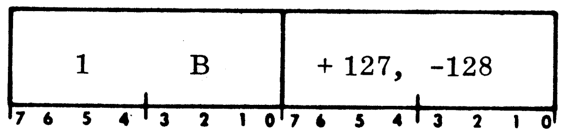
BS1 - Branch if Sense Switch 1 Set



The state of the sense switch 1 is tested and if it is set, a branch is made. If sense switch 1 is not set, no branch is made and the next sequential instruction is initiated.

If SSW1=1, (PC)+b→(PC)

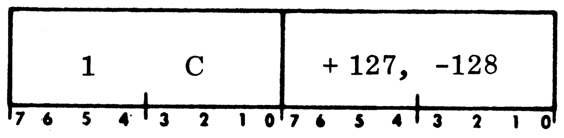
BS2 - Branch if Sense Switch 2 Set



The state of the sense switch 2 is tested and if it is set, a branch is made. If sense switch 2 is not set, no branch is made and the next sequential instruction is initiated.

If SSW2=1, (PC)+b→(PC)

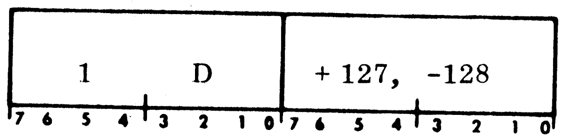
BS3 - Branch if Sense Switch 3 Set



The state of the sense switch 3 is tested and if it is set, a branch is made. If sense switch 3 is not set, no branch is made and the next sequential instruction is initiated.

If SSW3=1, (PC)+b→(PC)

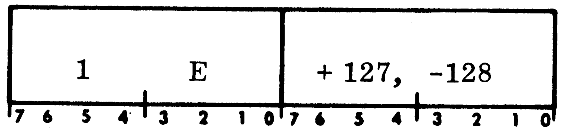
BS4 - Branch if Sense Switch 4 Set



The state of the sense switch 4 is tested and if it is set, a branch is made. If sense switch 4 is not set, no branch is made and the next sequential instruction is initiated.

If SSW4=1, (PC)+b→(PC)

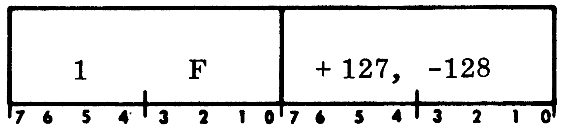
BTM - Branch on Teletype MARK



The state of the serial Teletype input line is checked. If the line is in the MARK or one condition, the branch is taken. Otherwise, the next instruction in sequence is initiated.

If MARK or one on TTY input line, (PC)+b→(PC)

BEP - Branch on Even Parity

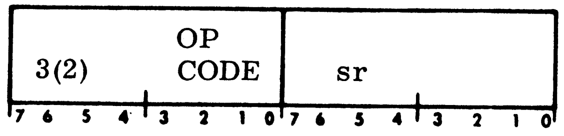


The state of the parity line from the central processor option board is checked. If the line indicates even parity, the branch is taken. Otherwise, the next instruction in sequence is executed. If the CPU option board is not installed, this instruction operates as an unconditional branch.

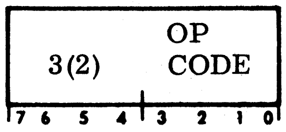
If even parity from CPU option board, (PC)+b→(PC)

SINGLE-REGISTER OPERATIONS

There are two types of single-register operations, explicit and implicit. Explicit register instructions are two-byte instructions with the source register address contained in the second byte. (See figure 2-1 for register designations.) The form of an explicit single-register instruction is:



Implicit register instructions are single-byte instructions with the source register implicit in the operation code. The form of an implicit single-register instruction is:

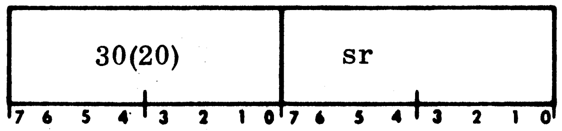


The least-significant byte (8 bits) of a 16-bit word is implied when using the byte mode.

The explicit instructions are described first, followed by the implicit instructions.

EXPLICIT REGISTER INSTRUCTIONS (TWO BYTES)

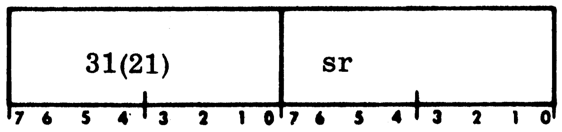
INR(B) - Increment Register



The contents of the source register is incremented by one.

$(sr)+1 \rightarrow (sr)$ Affected: sr, F, M, V

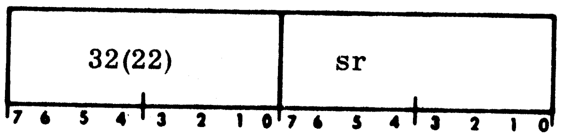
DCR(B) - Decrement Register



The contents of the source register is decremented by one.

$(sr)-1 \rightarrow (sr)$ Affected: sr, F, M, V

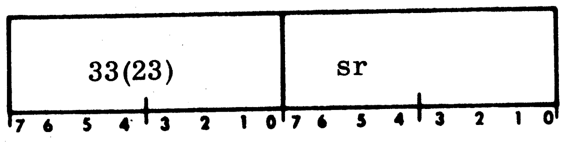
CLR(B) - Clear Register



The contents of the source register is cleared. All condition indicators except V are reset. V is set.

$1 \rightarrow (V)$ Affected: sr, F, L, M, V

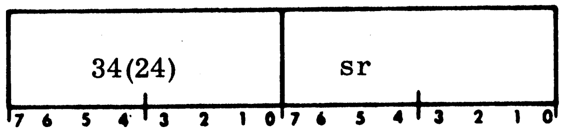
IVR(B) - Invert Register



The contents of the source register is set to the ones' complement.

$(\overline{sr}) \rightarrow (sr)$ Affected: sr, F, L, M, V

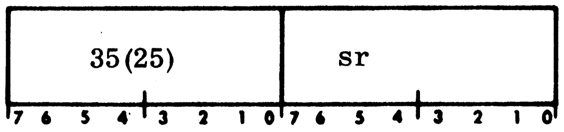
SRR(B) - Shift Register Right



The contents of the source register is shifted right one bit position. The least-significant bit of the register is shifted into the link condition flag, and the sign bit is extended.

$(sr)_{x+1} \rightarrow (sr)_x,\ (sr)_0 \rightarrow (L)$ Affected: sr, F, L, M, V

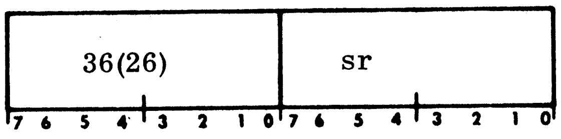
SLR(B) - Shift Register Left



The contents of the source register is shifted left one bit position. The most-significant bit of the register is shifted into the link condition flag and a zero is shifted into the least-significant bit.

$(sr)_x \rightarrow (sr)_{x+1},\ (sr)_{15(7)} \rightarrow (L),\ 0 \rightarrow (sr)_0$ Affected: sr, F, L, M, V

RRR(B) - Rotate Register Right

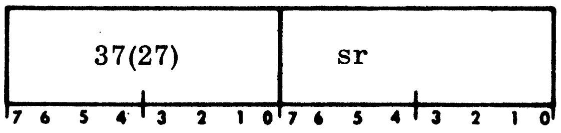


The contents of the source register is rotated right one bit position. The link condition flag is rotated into the most-significant bit position, and the least-significant bit of the register is rotated into the link flag.

$(sr)_{x+1} \rightarrow (sr)_{x}, \quad (sr)_{0} \rightarrow (L), \quad (L) \rightarrow (sr)_{15(7)}$

Affected: sr, F, L, M, V

RLR(B) - Rotate Register Left



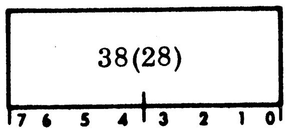
The contents of the source register is rotated left one bit position. The link condition flag is rotated into the least-significant bit position, and the most-significant bit of the register is rotated into the link flag.

$(sr)_{x} \rightarrow (sr)_{x+1}, \quad (sr)_{15(7)} \rightarrow (L), \quad (L) \rightarrow (sr)_{0}$

Affected: sr, F, M, L, V

IMPLICIT REGISTER INSTRUCTIONS (ONE BYTE)

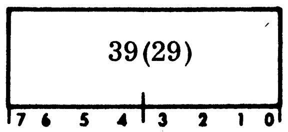
INA(B) - Increment Accumulator by 1



The accumulator is incremented by 1.

$(A)+1 \rightarrow (A)$ Affected: A, F, M, V

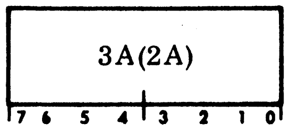
DCA(B) - Decrement Accumulator by 1



The accumulator is decremented by 1.

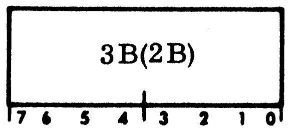
(A)-1→(A) Affected: A, F, M, V

CLA(B) - Clear Accumulator



The accumulator is cleared. All condition indicators except V are reset. V is set
1→(V) Affected: A, F, L, M, V

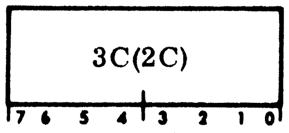
IVA(B) - Invert Accumulator



The accumulator is set to the one's complement.

$(\bar{A})$→(A) Affected: A, M, V

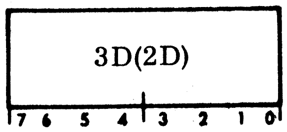
SRA(B) - Shift Accumulator Right



The contents of the accumulator is shifted right one bit position. The least-significant bit of the register is shifted into the link condition flag and the sign bit is extended.

$(A)_{x+1} \rightarrow (A)_x$, $(A)_0 \rightarrow (L)$ Affected: A, F, L, M, V

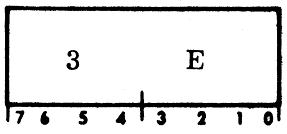
SLA(B) - Shift Accumulator Left



The contents of the accumulator is shifted left one bit position. The most-significant bit of the accumulator is shifted into the link condition flag and a zero is shifted into the least-significant bit.

$(A)_x \rightarrow (A)_{x+1}, (A)_{15(7)} \rightarrow (L), 0 \rightarrow (A)_0$ Affected: A, F, L, M, V

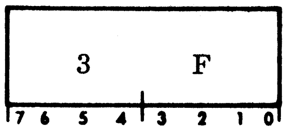
INX - Increment X Register



The X register (16 bits) is incremented by one.

$(X)+1 \rightarrow (X)$ Affected: X, F, M, V

DCX - Decrement X Register

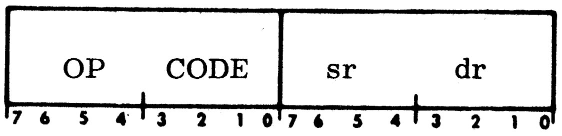


The X register (16 bits) is decremented by one.

$(X)-1 \rightarrow (X)$ Affected: X, F, M, V

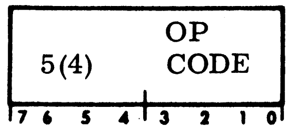
DOUBLE REGISTER OPERATIONS

There are two types of double-register operations, explicit and implicit. Explicit register instructions are two-byte instructions with the source register and destination register addresses contained in the second byte. The form of an explicit double-register instruction is:



The operation code may specify either a byte or full word operation. In the byte mode, the least-significant byte is assumed. The source register and destination register may be the same. See figure 2-1 for register designations.

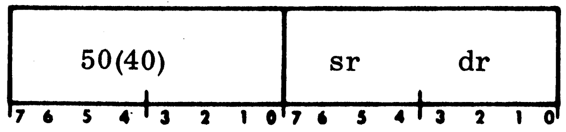
Implicit register instructions are single-byte instructions with the source and destination registers implicit in the operation code. The form of an implicit double-register instruction is:



The explicit instructions are described first, followed by the implicit instructions.

EXPLICIT REGISTER INSTRUCTIONS (TWO BYTES)

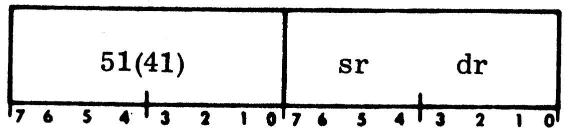
ADD(B) - Add



The contents of the source register and the contents of the destination register are added together. The sum is deposited in the destination register.

$(dr)+(sr) \rightarrow (dr)$ Affected: dr, F, L, M, V

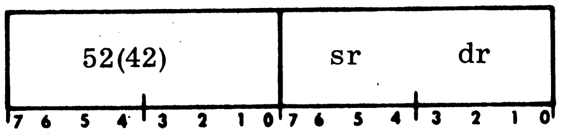
SUB(B) - Subtract



The contents of the source register minus the contents of the destination register is deposited in the destination register.

$(sr)-(dr) \rightarrow (dr)$ Affected: dr, F, L, M, V

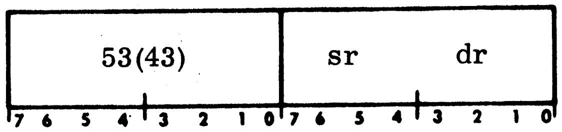
AND(B) - AND



The logical product of the source register and the destination register is deposited in the destination register.

$(dr) \wedge (sr) \rightarrow (dr)$ Affected: dr, M, V

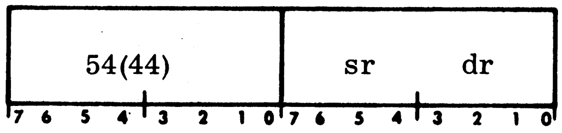
ORI(B) - OR Inclusive



The logical sum (OR) of the source register and the destination register is placed in the destination register.

$(dr) \vee (sr) \rightarrow (dr)$ Affected: dr, M, V

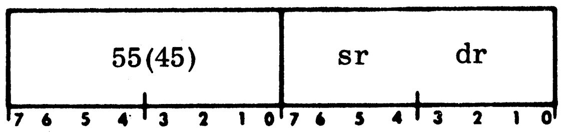
ORE(B) - OR Exclusive



The exclusive OR, or inequality function of the bits of the source register, and the destination register is deposited in the destination register.

$(dr) \underline{\vee} (sr) \rightarrow (dr)$ Affected: dr, M, V

XFR(B) - Transfer

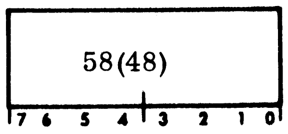


The contents of the source register is deposited in the destination register. The source register is unaltered. The original contents of the destination register is lost.

(sr)→(dr) Affected: dr, M, V

IMPLICIT REGISTER INSTRUCTIONS (ONE BYTE)

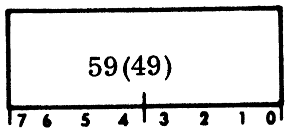
<u>AAB(B) - Add A Register and B Register</u>



The contents of the A register and the contents of the B register are added together. The sum is deposited in the B register.

(A)+(B)→(B) Affected: B, F, L, M, V

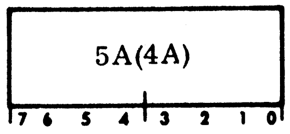
<u>SAB(B) - Subtract A Register and B Register</u>



The contents of the A register minus the contents of the B register is deposited in the B register.

(A)-(B)→(B) Affected: B, F, L, M, V

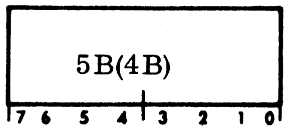
<u>NAB(B) - AND A Register and B Register</u>



The logical product of the A register and the B register is deposited in the B register.

(B) $\wedge$ (A) → (B) Affected: B, F, M, V

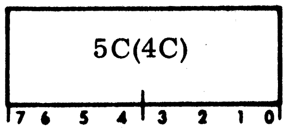
XAX(B) - Transfer A Register to X Register



The contents of the A register is deposited in the X register. The A register is unaltered. The original contents of the X register are lost.

(A)→(X) Affected: X, M, V

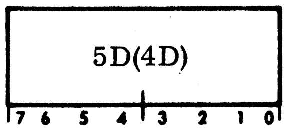
XAY(B) - Transfer A Register to Y Register



The contents of the A register is transferred to the Y register. The A register is unaltered. The original contents of the Y register are lost.

(A)→(Y) Affected: Y, M, V

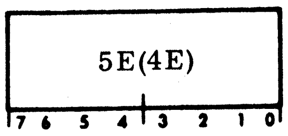
XAB(B) - Transfer A Register to B Register



The contents of the A register is deposited in the B register. The A register is unaltered. The original contents of the B register are lost.

(A)→(B) Affected: B, M, V

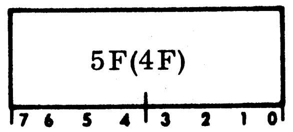
XAZ(B) - Transfer A Register to Z Register



The contents of the A register is transferred to the Z register. The A register is unaltered. The original contents of the Z register are lost.

(A)→(Z) Affected: Z, M, V

XAS(B) - Transfer A Register to S Register

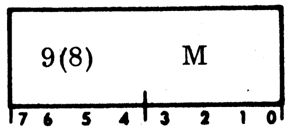


The contents of the A register is transferred to the S register. The A register is unaltered. The original contents of the S register are lost.

(A)→(S) Affected: S, M, V

MEMORY REFERENCE INSTRUCTIONS

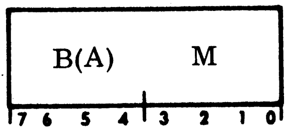
LDA(B) - Load A Register



The load A register instruction references an operand located in memory and deposits it in the A register. The instruction may be one byte (register addressing), two bytes (relative addressing), or three bytes (direct addressing).

$(EA) \rightarrow (A)$ Affected: A, F, L, M, V

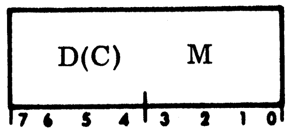
STA(B) - Store A Register



The store A register instruction takes the contents of the A register and deposits it in memory. The instruction may be one byte (register addressing), two bytes (relative addressing), or three bytes (direct addressing).

$(A) \rightarrow (EA)$ Affected: (EA), F, L, M, V

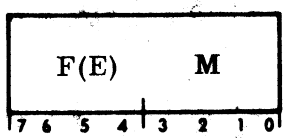
LDB(B) - Load B Register



The load B register instruction references an operand located in memory and deposits it in the B register. The instruction may be one byte (register addressing), two bytes (relative addressing), or three bytes (direct addressing).

$(EA) \rightarrow (B)$ Affected: B, F, L, M, V

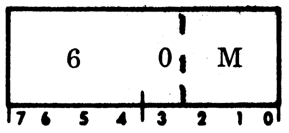
STB(B) - Store B Register



The store B register instruction takes the contents of the B register and deposits it in memory. The instruction may be one byte (register addressing), two bytes (relative addressing), or three bytes (direct addressing).

(B) →(EA) Affected: (EA), F, L, M, V

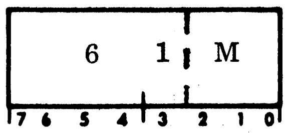
LDX - Load X Register Word



Load X register (word) instruction references a 16-bit operand (always 16-bit) located in memory and deposits it in the 16-bit file register designated X. M can be 0 through 5.

(EA)→ (X) Affected: X, F, L, M, V

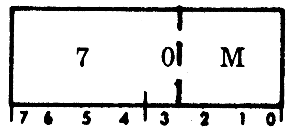
STX - Store X Register Word



Store X register (word) deposits the X file register (always 16-bits) into memory. M can be 0 through 5.

(X) →(EA) Affected: (EA), F, L, M, V

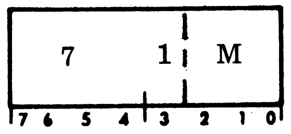
JMP - Unconditional Jump



The unconditional jump instruction takes the operand address and deposits it in the program counter. M can be 1 through 5.

EA→(PC)   Affected: None

<u>JSR - Jump to Subroutine</u>



A jump to a subroutine instruction causes an alternative to the program counter similar to an unconditional jump. The value loaded into the program counter is arrived at via address modification in just the same way as the JMP instruction. Before this alteration of the program counter takes place, however, certain other activities are performed.

The current program counter contents must be preserved so that a return from the subroutine can be effected. This is accomplished automatically by the hardware. The contents of the X register (16 bits) is stored in memory at the locations specified by the S register (files A, B). The S register is decremented by two prior to storing and left pointing at the 16-bit operand. The current program counter contents is stored in the X register. The effective address is placed in PC and in P. M can be 1 through 5.

(X)↓, (PC)→(X), EA→(PC), EA→(P)   Affected: P, S, X

4. MEMORY AND INPUT/OUTPUT CHARACTERISTICS

INTRODUCTION

The interface point for memory modules and input/output device controllers (IODC's) is the same Universal bus. The central processor uses the bus to communicate with memory locations or with I/O controllers. For all load and store operations an I/O controller is treated like a memory location with an address equal to or greater than $61,440_{10}$ ($F000_{16}$).

DATA AND ADDRESS LINES

DATA BUS

The data bus consists of eight lines which are designated $\overline{DB00}$ through $\overline{DB07}$. These lines represent a data byte to or from either memory or an IODC. The logic levels are:

0 v $\pm$ 0.45 v = logical 1

+3 v to +5 v = logical 0

ADDRESS BUS

The address bus consists of 16 lines which are designated $\overline{AB00}$ through $\overline{AB15}$. These lines represent either a memory address (0-61,339) or an IODC address (61,440-65,535). The logic levels are:

0 v $\pm$ 0.45 v = logical 1

+3 v to +5 v = logical 0

CONTROL LINES

The control lines used in the memory and I/O interface are described, mnemonics given, and source identified in table 4-1.

Table 4-1. Control Lines

Signal	Mnemonic	Source			Meaning
		CPU	MEM	I/O	
SYNCHRONOUS DEVICE	DSYN			X	This line is enabled by all standard memory modules and some I/O controllers. Ready response of this module will be synchronous with the CPU clock and saves a clock cycle.
I/O REQUEST	IORQ			X	This line is activated by an I/O device wishing to do an automatic transfer to/from memory or operational registers.
I/O REQUEST	INTR			X	This line is activated by an I/O controller wishing to do an interrupt.
I/O ACKNOWLEDGE	IACK	X			CPU response to IORQ when it has reached a point where it can release the bus for use by an I/O controller.
I/O DONE	IDON			X	This line is activated by an I/O device when it releases the bus back to the computer.
I/O NOT BUSY	IOBY	X			CPU response to IDON. Used by I/O controller to turn off IDON.
SERIAL I/O PRIORITY	INPO/ INPI			X	Serial chain for I/O device controllers to establish priority during competitive interrupt requests.

Signal	Mnemonic	Source			Meaning
		CPU	MEM	I/O	
SERIAL PRIORITY LINE FOR AUTOMATIC I/O	IOPO/ IOPI			X	Serial chain for I/O device controllers to establish priority during competitive requests for automatic I/O.
INITIATE READ	RDIN	X		X	Read Initiate from the CPU starts a memory read cycle or an I/O transfer to the accumulator. Read Initiate from an I/O controller occurs during automatic block I/O.
INITIATE WRITE	WTIN	X		X	Write Initiate from the CPU starts a memory write cycle or an I/O transfer from the accumulator. Write Initiate from an I/O controller occurs during automatic block I/O.
MEMORY OR I/O BUSY	BUSY		X	X	Busy from a memory or I/O module will arrest the CPU at the point where it is about to change address lines on the bus. Allows synchronous memory operations and closed-loop I/O.
DATA READY	DRDY		X	X	Data Ready from a memory or I/O module indicates the completion of the read access time. The CPU or I/O controller pauses and waits for this line before sampling the data bus.

Signal	Mnemonic	Source			Meaning
		CPU	MEM	I/O	
INTERRUPT REQUEST	$\overline{\text{INR3}}$ $\overline{\text{INR2}}$ $\overline{\text{INR1}}$ $\overline{\text{INR0}}$			X	These lines are activated by an I/O interrupting device when its priority level is higher than the current level and it wishes the CPU to take some immediate action. The CPU executes a level change to a new register set at the base location supplied on the encoded lines. For example, a code of 0100 causes a level change to the register set at location 40_{16}.
MASTER RESET	MRST	X			This line is used to initialize all I/O device controllers.
ADDRESS PRE-EMPTED	APRE		X		High priority memory module indicates it has an address the same as another memory module. The second memory module should ignore the address; e.g., in basic, op registers are mapped into 0-0F of main memory. When high-speed file of 16 bytes is inserted, it assumes addresses 0-0F and should disable main memory for those addresses. This line is also used to enable/disable parity generation and checking when the CPU option board is in the system. When this line is enabled, parity generation and checking is disabled.

Signal	Mnemonic	Source			Meaning
		CPU	MEM	I/O	
CURRENT LEVEL	$\overline{CL3}$ $\overline{CL2}$ $\overline{CL1}$ $\overline{CL0}$	X			Indicates current processor priority level.

OPTIONAL LINES

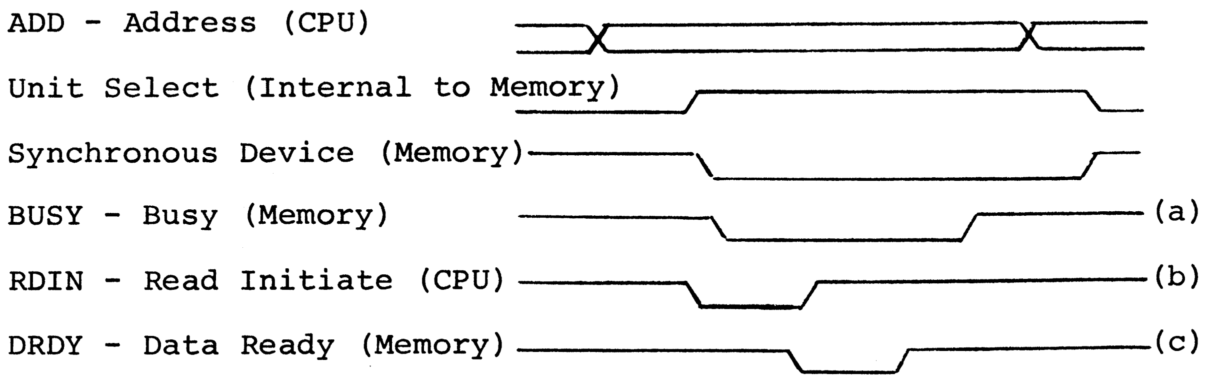
PARITY DATA BIT - MD08

A ninth memory or I/O data bit is provided for the optional parity generation, check feature. This bit is active for any memory or I/O transfer when the parity option is installed.

PARITY ERROR STATUS LINE

When the optional parity checker detects a parity error on a memory read, or an I/O controller detects a parity error on a data byte transfer, this status line causes a program interruption to a programmable interrupt level.

MEMORY CYCLE


ADD - Address (CPU)
Unit Select (Internal to Memory)
Synchronous Device (Memory)
BUSY - Busy (Memory)
RDIN - Read Initiate (CPU)
DRDY - Data Ready (Memory)
(a)
(b)
(c)

(a) Turns off when internal memory cycle is complete.

(b) Turns off when data ready turns on.

(c) Determined by memory access time.

The beginning of a memory cycle is defined by the Read Initiate (or Write Initiate) line. Prior to this the address must have been applied to the bus and the bus given time to settle. This is accomplished by delaying one CPU clock time from address to Read Initiate.

Read Initiate starts the selected memory module through a read cycle in nondestructive memory or through a Read/Restore cycle if destructive.

Data Ready identifies the point when data access from memory storage is complete and data is on the bus.

Busy is generated by the memory module to identify the total cycle time. It prevents the CPU from initiating a new cycle or changing the memory address until the restore operation has been performed (as for example in a magnetic core memory).

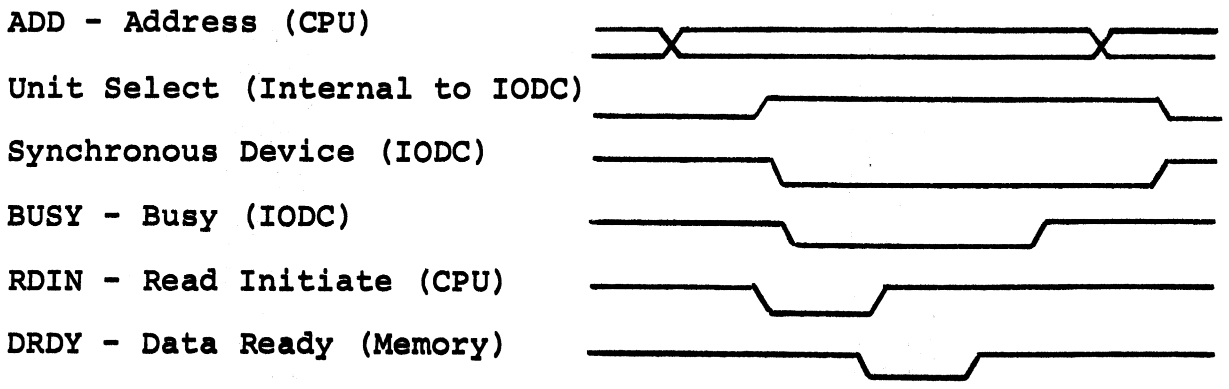
For high-speed files or some ROM's, Busy may go "off" with Data Ready going "on" to signify that the cycle is complete at the same time that the data access is complete.

Device Synchronous indicates to the CPU that the return of "Data Ready" is synchronous to the CPU clock. The CPU loads the memory bus data on the next clock pulse, thereby eliminating the time lost in synchronizing the Data Ready line.

A memory module need not use the line, but for minimum access time it is recommended.

INPUT/OUTPUT CYCLE

The basic criterion for input/output device controllers is that they look like memory modules to the CPU. The CPU instructions Read I/O, Write I/O are similar to the memory Load and Store instructions. The only real difference is the address value.


ADD - Address (CPU)
Unit Select (Internal to IODC)
Synchronous Device (IODC)
BUSY - Busy (IODC)
RDIN - Read Initiate (CPU)
DRDY - Data Ready (Memory)

The timing is similar to a memory cycle with the IODC acting as one or more 8-bit storage registers addressable by the CPU.

CAPACITY

The total mix of memory modules and I/O device controllers in the basic chassis is limited to 12. These can be divided between them in any mix.

The expansion capacity is limited by the number of electrical sources and loads each bus line can accommodate.

To expand this number of attachments, an interface expander board is used. This expander adds a driver-receiver pair to each line and uses one of the available loads. The expander allows an additional 10 devices (memory or I/O controllers) to be added. For highest instruction execution rate, memory modules should be placed before the expander.

INPUT/OUTPUT DEVICE CONTROLLERS

The input/output device controllers (IODC's) for the ee 200 consist of an I/O bus interface and control section and a machine control section.

The I/O bus interface and control sections are of two types: (1) those that handle transfers under program control only; and (2) those that include automatic block transfer capability. Within these two categories, the design is similar for all controllers.

The machine control circuits are different for each device to be controlled. They vary from simple registers and multiplexers for digital input/output interface boards to long sequential controllers and registers for more sophisticated machines; e.g., magnetic tape units.

PROGRAM CONTROL

Program control of an IODC means total software manipulation of a data transfer or control function. In general, it requires the following:

+ Execution of a control instruction to alert the device to get ready for transfer; e.g., start paper tape reader.

+ Execution of an instruction to determine device readiness to transfer a piece of data; e.g., test or sense status.

+ Execution of an instruction which will transfer the data unequivocally; i.e., open-loop.

AUTOMATIC BLOCK TRANSFER

Automatic block transfer capability significantly reduces software overhead. This capability is achieved by adding address generating and memory control capability to the basic hardware. The address generator is a register which is loaded via a program control transfer with the starting address of the block to be transferred. When the machine control is ready, a data byte is automatically transferred into or out of memory at the desired address. The address is updated as each transfer occurs until the end address is reached. At this time, the process is terminated.

A device coupler with automatic block transfer capability also has program transfer capability.

DEVICE ADDRESS ASSIGNMENT

The address to an IODC is 16 bits in length. Bits 12 through 15 form the I/O flag and are all "ones" for input/output operations. Bits 0 through 11 are the device address and function expansion bits.

Bits 10 and 11 of the device address can be changed by installing jumpers on the IODC board. This enables the use of more than one of the same type peripheral device in a single system.

The 12-bit address field provides a total of 4,096 device addresses. The address plus the output data byte (or word) allows 24 or 32 bits for control.

SERIAL TELETYPE

Included in the basic hardware is a teletype interface for an ASR 33 or 35 Teletype. This serial channel is not a part of the bus system but resides instead inside the CPU.

The majority of the work in using this interface is in the program.

The Enable Link Out instruction (ELO) gates the content of the link flag to the serial output channel, thereby sending a MARK (1) or SPACE (0) to the Teletype. The processor program then times out the required period and changes to the next bit of the code to be sent.

When completed, the line must be returned to the MARK (1) condition which is the quiescent state for teletypes.

The Branch on Teletype MARK instruction (BTM) is dedicated to detecting the state of the serial input line from the Teletype.

To monitor for the initial MARK to SPACE transition, the computer must constantly examine the line. When it detects the initial transition it uses a timing loop similar to the output operation to mark off a bit period before sampling the line.

All the timing, searching, assembly for input; as well as the disassembly and timing for output are accomplished by software.

5. INTERRUPT STRUCTURE

INTRODUCTION

The ee 200 Computer was designed for real-time systems applications. This requires the system to be responsive to a variety of external stimuli such as external alarms, end-of-block signals for I/O device controllers, timing flags, etc. The frequency and urgency of these conditions are not always related. For example, a power-loss signal from the power supply seldom occurs but has extreme urgency. A real-time clock interrupt occurs frequently, but servicing it is a nonurgent requirement.

To accommodate this range of frequency and urgency requires rapid attention by the processor; i.e., low overhead time in arresting the current process and transferring to the interrupt handling subroutine, and a priority scheme where conflicting interrupts are serviced in the proper order.

EXTERNAL INTERRUPT LINES

When an interrupting source desires attention, it first examines the Current Level Status lines (CLO-CL3) from the CPU. If the current processor level is below that of the requesting device, it supplies an Interrupt Request Line (INRQ). When the request is acknowledged by the processor (INAK), the interrupting device supplies a 4-bit code (1-F) on lines INRO-INR3. Upon completion of the current instruction, the processor automatically jumps to a routine whose address is stored in the P register at the level defined by the encoded lines.

Table 5-1 shows the memory locations associated with each interrupt level. These locations must be initialized prior to servicing interrupt requests.

Table 5-1. Interrupt Level Memory Addresses

Interrupt Line Code	Corresponding Interrupt Level	Subroutine Address Stored in Memory Locations (P Register)
0001	IL01	1E, 1F
0010	IL02	2E, 2F
0011	IL03	3E, 3F
0100	IL04	4E, 4F
0101	IL05	5E, 5F
0110	IL06	6E, 6F
0111	IL07	7E, 7F
1000	IL08	8E, 8F
1001	IL09	9E, 9F
1010	IL0A	AE, AF
1011	IL0B	BE, BF
1100	IL0C	CE, CF
1101	IL0D	DE, DF
1110	IL0E	EE, EF
1111	IL0F	FE, FF

The 15 levels have attendant priorities; IL0F is the highest and IL01 is the lowest. During competitive situations, a higher priority level will always gain access ahead of a lower priority level. If entry has already been made to a lower priority handling routine, the higher level can immediately draw the processor out of the lower routine to service the higher level. When the higher level is complete, a linkage is available to allow the processor to thread back to the lower level program. A similar capability can often be claimed for other machines but it is typically effected with significant software overhead. This software overhead manifests itself in real time during which the machine is nonresponsive to higher levels. In the ee 200 Computer this time is never more than one instruction execution time.

There are no noninterruptible instruction sequences. A HLT instruction is interruptible. An interrupt moves the machine to the Run State at the interrupt level program.

DYNAMIC OPERATIONAL REGISTER ALLOCATION

Handling a real time environment involves rapid context switching; i.e., moving from one process to another with a minimum of overhead in time and instructions.

The usual approach is to allow a jump and save instruction to store the current program counter value. Switching of flags (fault, zero, etc.) and storing of current operands requires significant hardware or several software steps (which must be noninterruptible).

In the ee 200 Computer, this entire process is eliminated. Context switching (the changing of processor environments) is accomplished entirely automatically with no need to save and restore register values and flags.

The operational registers, accumulators, index registers, etc., are addressable in memory locations. Normal background processes are conducted with the operational registers residing in memory locations 0 through F (the first 16 bytes). Interrupt levels which are higher than the background process in priority allow the register definition to be reallocated to successive locations in memory.

The actual location is related to interrupt line number. IL01 will cause relocation of operational registers to locations 10-1F; IL0F causes reallocation to F0-FF.

Each time a new allocation is made, a link to the old level is saved. In this manner higher priority interrupts cause immediate context switching with the ability to thread back through the lower processes when the high one is complete. Eventually this linkage will cause the register allocation to return to 0-F, the normal background registers.

Figure 5-1 shows the register allocation by interrupt level.

PRIORITY WITHIN A LEVEL

More than one interrupting device may share an interrupt level. When an interrupt occurs on a shared level, the CPU polls the sharing devices to determine which one initiated the interrupt.

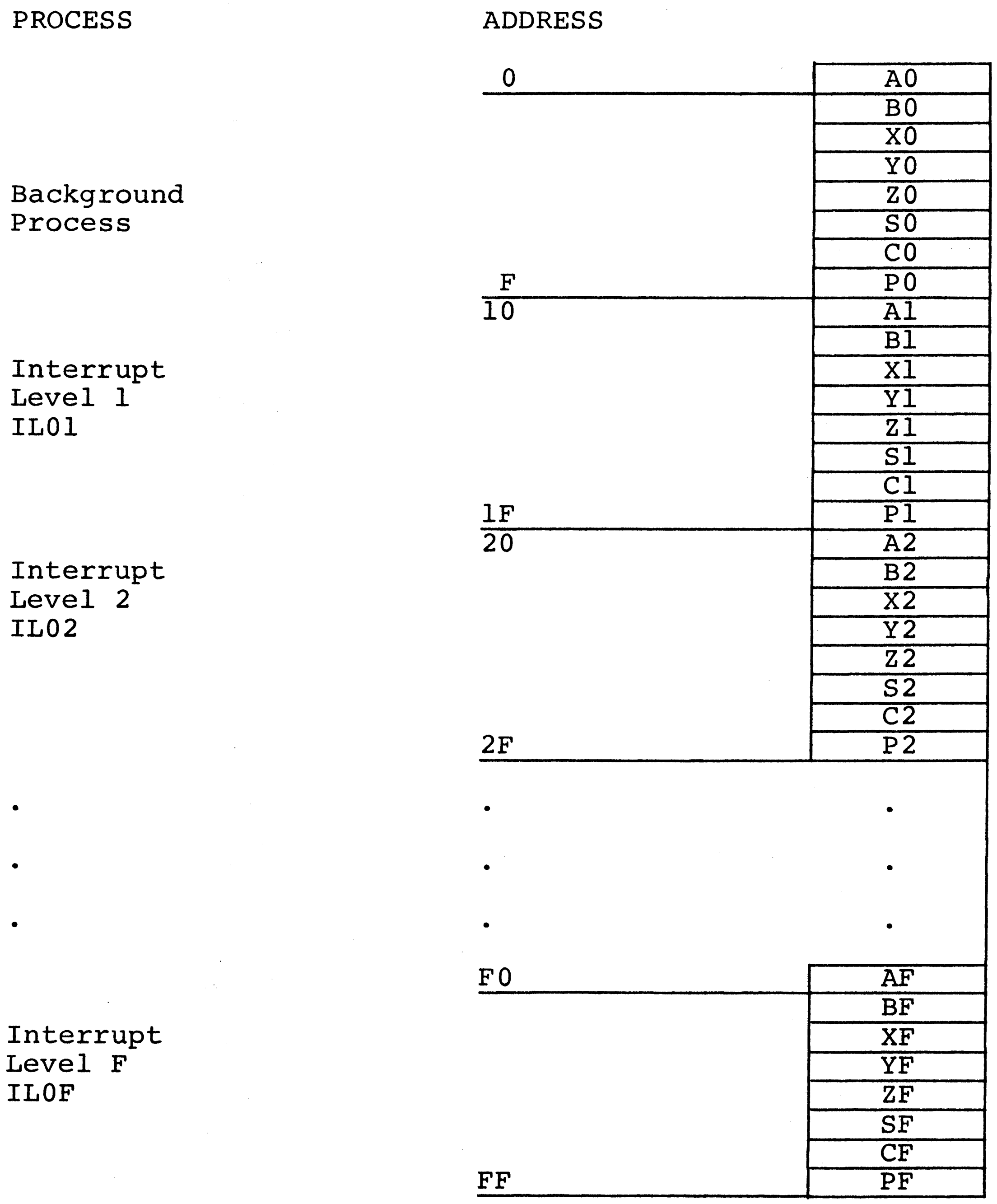


Figure 5-1. Register Allocation by Interrupt Level

6. CONTROLS AND INDICATORS

BASIC CONTROL PANEL

The basic control panel, shown in figure 6-1, connects directly into the CPU and controls processor fundamentals such as RUN, HLT, RESET, etc. The basic control panel controls and indicators are described in the following paragraphs.

SENSE Switches

There are four two-position sense switches located on the control panel. The switches are designated 1 through 4. The state of each switch is sensed by a branch instruction. If a switch is ON (DOWN) when sensed, a branch is made. If a switch is OFF (UP) when sensed, the next instruction in sequence is executed.

RUN Switch and Indicator

The RUN switch is a pushbutton switch. Pressing RUN puts the computer in the RUN mode and program execution begins. The RUN indicator is located immediately above the switch and is illuminated when the computer is in the RUN mode.

HALT Switch and Indicator

The HALT switch is a pushbutton switch. If the computer is in the RUN mode and HALT is pressed, the computer will go to the HALT (WAIT) mode at the end of the current instruction. If the computer is in the HALT mode and HALT is pressed, a single instruction is executed. The HALT indicator is located immediately above the switch and is illuminated when the computer is in the HALT mode.

RESET Switch

The RESET switch is a pushbutton switch. Pressing RESET initializes all CPU and input/output device coupler logic.

LOAD Switch

The LOAD switch is a pushbutton switch. Pressing LOAD forces the CPU to the RUN state and initiates program loading from the device as specified in a ROM loader program.

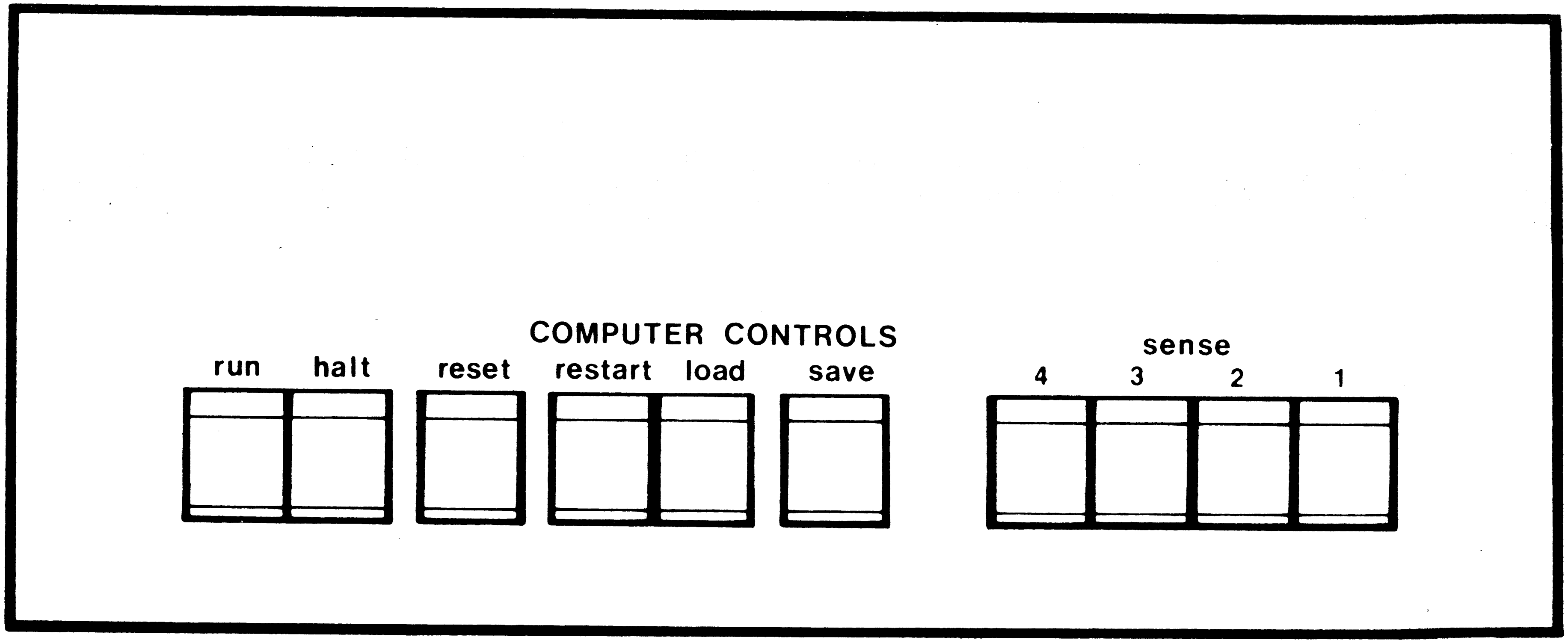


Figure 6-1. Computer Controls

RESTART

The RESTART switch is a pushbutton switch. Pressing RESTART forces the CPU to the RUN state with an initial address determined by sense switches 1 and 2. Switch 1 causes the start address to be taken from $1A_{16}$ and $1B_{16}$. Switch 2 causes the address to be taken from $2A_{16}$ and $2B_{16}$. Switches 1 and 2 will refer to $3A_{16}$ and $3B_{16}$.

LINK Indicator

The LINK indicator is ON when the link condition flip-flop is set and is OFF when the flip-flop is reset.

OVERFLOW Indicator

The OVERFLOW indicator is ON when the fault condition flip-flop is set and is OFF when the fault condition flip-flop is reset.

TTY LOAD

Depression of TTY Load switch causes automatic load from the Teletype directly via the CPU.

7. PACKAGING TECHNIQUES

INTRODUCTION

This section describes the packaging techniques used in constructing the ee 200 Computer.

MECHANICAL CONFIGURATION

The basic computer housing is a rack-mounted chassis with dimensions as follows:

Height:	8.75 inches
Width:	19 inches
Depth:	18.5 inches

The printed-circuit boards are mounted vertically and plug in from the rear of the chassis. Circuit board dimensions are:

Height:	8 inches
Length:	14.3 inches

Each circuit board has one connector on the front edge of the board (P1) and either one or two connectors on the rear (P2 and P3). Connector P1 plugs into the input/output memory bus. Connectors P2 and P3 (if required) are used for interboard connections and I/O device connection.

CARD SLOT DEDICATION

When viewing the chassis from the card entry view, card slots are numbered 1 through 16, from left to right. Card slot assignments are:

Slot	Usage
1-3	CPU (Control Board 1, Control Board 2, Data Board)
4	ROM Loader
5-16	High-speed register files, MOS main memory, core main memory, all peripheral device controllers

POWER SUPPLY

The self-cooled power supply is of modular construction and is mounted separately from the computer to allow customizing for expanded systems. Connection to the computer is via a cable of up to 5 feet in length. Remote sensing of the supplied voltages is provided. The ac input is 115/230 v ac, ±10%, 47-63 Hz. The power supply dimensions are:

Height:	6.5	inches
Width:	11	inches
Depth:	6	inches

APPENDIX A

INSTRUCTION LIST

OP CODE	MNEMONIC	NAME	OP STATEMENT	CONDITION FLAGS F	L	M	V
Control (1 Byte)							
00	HLT	Wait for Interrupt (Halt)	(PC)→(P), 0→(R)	-	-	-	-
01	NOP	No Operation	None	-	-	-	-
02	SF	Set Fault	1→(F)	1	-	-	-
03	RF	Reset Fault	0→(F)	0	-	-	-
04	EI	Enable Interrupt System	1→(I)	-	-	-	-
05	DI	Disable Interrupt System	0→(I)	-	-	-	-
06	SL	Set Link	1→(L)	-	1	-	-
07	RL	Reset Link	0→(L)	-	0	-	-
08	CL	Complement Link	$(\overline{L})$→(L)	-	c	-	-
09	RSR	Return from Subroutine	(X)→(PC), ↑(X)	-	-	-	-
0A	RI	Return from Interrupt	(PC)→(P), Status→C_{12-15}, C_{4-7}→LV	c	c	c	c
0B	RIM	Return from Interrupt Modified	Status→C_{12-15}, C_{4-7}→LV	c	c	c	c
0C	ELO	Enable Link Out	(L)→Serial TTY Interface	-	-	-	-
0D	PCX	Transfer PC to X	(PC)→(X)	-	-	-	-
0E	DLY	Delay 4.55 milliseconds	Delay 4.55 milliseconds	-	-	-	-
Conditional Branches (2 Bytes)			If Condition True, (PC)+b→(PC). Otherwise, continue with next Instruction.				
10	BL	Branch if Link Set	(L)=1	-	-	-	-
11	BNL	Branch if Link Not Set	(L)=0	-	-	-	-
12	BF	Branch if Fault Set	(F)=1	-	-	-	-
13	BNF	Branch if Fault Not Set	(F)=0	-	-	-	-
14	BZ	Branch if Equal to Zero	(V)=1	-	-	-	-
15	BNZ	Branch if Not Equal to Zero	(V)=0	-	-	-	-
16	BM	Branch if Minus Set	(M)=1	-	-	-	-
17	BP	Branch on Positive	(M)=0	-	-	-	-
18	BGZ	Branch if Greater than Zero	(M) $\wedge$ (V) = 0	-	-	-	-
19	BLE	Branch if Less Than or Equal to Zero	(M) $\vee$ (V) = 1	-	-	-	-
1A	BS1	Branch if Sense Switch 1 Set	SSW1=1	-	-	-	-
1B	BS2	Branch if Sense Switch 2 Set	SSW2=1	-	-	-	-
1C	BS3	Branch if Sense Switch 3 Set	SSW3=1	-	-	-	-
1D	BS4	Branch if Sense Switch 4 Set	SSW4=1	-	-	-	-
1E	BTM	Branch on Teletype MARK	MARK on TTY Input Line	-	-	-	-
1F	BEP	Branch on Even Parity	If CPU Option Board In and Parity Even. If No Option Board, Branch Unconditional.	-	-	-	-
Single Register Operations (1 or 2 Bytes)							
30(20)	INR(B)	Increment Register by 1	(sr)+1→(sr)	c	-	c	c
31(21)	DCR(B)	Decrement Register by 1	(sr)-1→(sr)	c	-	c	c
32(22)	CLR(B)	Clear Register	0's→(sr)	0	0	0	1
33(23)	IVR(B)	Invert Register (1's complement)	$(\overline{sr})$→(sr)	-	-	c	c
34(24)	SRR(B)	Shift Right	$(sr)_{x+1}$→$(sr)_x$, $(sr)_0$→(L)	-	c	c	c
35(25)	SLR(B)	Shift Left	$(sr)_x$→$(sr)_{x+1}$, $(sr)_{15(7)}$→(L), 0→$(sr)_0$	c	c	c	c
36(26)	RRR(B)	Rotate Right	$(sr)_{x+1}$→$(sr)_x$, $(sr)_0$→(L), (L)→$(sr)_{15(7)}$	-	c	c	c
37(27)	RLR(B)	Rotate Left	$(sr)_x$→$(sr)_{x+1}$, $(sr)_{15(7)}$→(L), (L)→$(sr)_0$	c	c	c	c
38(28)	INA(B)	Increment A by 1	(A)+1→(A)	c	-	c	c
39(29)	DCA(B)	Decrement A by 1	(A)-1→(A)	c	-	c	c
3A(2A)	CLA(B)	Clear A	0→(A)	0	0	0	1
3B(2B)	IVA(B)	Invert A	$(\overline{A})$→(A)	-	-	c	c
3C(2C)	SRA(B)	Shift Right A	$(A)_{x+1}$→$(A)_x$, $(A)_0$→(L)	-	c	c	c
3D(2D)	SLA(B)	Shift Left A	$(A)_x$→$(A)_{x+1}$, $(A)_{15(7)}$→(L), 0→$(A)_0$	c	c	c	c
3E	INX	Increment X (Word) by 1	(X)+1→(X)	c	-	c	c
3F	DCX	Decrement X (Word) by 1	(X)-1→(X)	c	-	c	c

APPENDIX A

INSTURCTION LIST (CONT'D)

OP CODE	MNEMONIC	NAME	OP STATEMENT	CONDITION FLAGS F L M V
Double Register Operations (1 or 2 Bytes)				
50(40)	ADD(B)	Add	(dr)+(sr)→(dr)	c c c c
51(41)	SUB(B)	Subtract	(sr)-(dr)→(dr)	c c c c
52(42)	AND(B)	AND	(dr) ∧ (sr) → (dr)	- - c c
53(43)	ORI(B)	OR Inclusive	(dr) V (sr) → (dr)	- - c c
54(44)	ORE(B)	OR Exclusive	(dr) ⊻ (sr) → (dr)	- - c c
55(45)	XFR(B)	Transfer	(sr)→(dr)	- - c c
58(48)	AAB(B)	Add A Register and B Register	(B)+(A)→(B)	c c c c
59(49)	SAB(B)	Subtract A Register and B Register	(A)-(B)→(B)	c c c c
5A(4A)	NAB(B)	AND A Register and B Register	(B) ∧ (A) → (B)	- - c c
5B(4B)	XAX(B)	Transfer A Register to X Register	(A)→(X)	- - c c
5C(4C)	XAY(B)	Transfer A Register to Y Register	(A)→(Y)	- - c c
5D(4D)	XAB(B)	Transfer A Register to B Register	(A)→(B)	- - c c
5E(4E)	XAZ(B)	Transfer A Register to Z Register	(A)→(Z)	- - c c
5F(4F)	XAS(B)	Transfer A Register to S Register	(A)→(S)	- - c c
Memory Reference (1, 2 or 3 Bytes)				
90(80)	LDA(B)	Load A Register	(EA)→(A)	- - c c
B0(A0)	STA(B)	Store A Register	(A)→(EA)	- - c c
D0(C0)	LDB(B)	Load B Register	(EA)→(B)	- - c c
F0(E0)	STB(B)	Store B Register	(B)→(EA)	- - c c
60	LDX	Load X Register	(EA)→(X)	- - c c
68	STX	Store X Register	(X)→(EA)	- - c c
70	JMP	Jump	(EA)→(PC)	- - - -
78	JSR	Jump to Subroutine	(X)↓, (PC)→(X), EA→(PC), EA→(P)	- - - -

NOTES

1. When an instruction can be used in either the byte or the word mode, the hexadecimal operation code for the word mode is given first, followed by the byte operation code in parenthesis.

2. Instructions which can be used in either the byte or the word mode are indicated by a "B" enclosed in parenthesis immediately following the instruction mnemonic.

3. Notation used in the operation statements is described in section two.

4. Condition flag notation is as follows:

   0 = Reset
   1 = Set
   c = Conditionally Set/Reset
   - = Not Affected

APPENDIX B

HEXADECIMAL/DECIMAL INTEGERS

Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4,294,967,296	1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	8,589,934,592	2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	12,884,901,888	3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	17,179,869,184	4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	21,474,836,480	5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	25,769,803,776	6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	30,064,771,072	7	1,879,048,192	7	177,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	34,359,738,368	8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	38,654,705,664	9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	42,949,672,960	A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	47,244,640,256	B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	51,539,607,552	C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	55,834,574,848	D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	60,129,542,144	E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	64,424,509,440	F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
	9		8		7		6		5		4		3		2		1

HEXADECIMAL TO DECIMAL

This table allows for conversion of hexadecimal numbers of up to nine characters in length to their decimal equivalents.

Locate the columns in the table corresponding to the position of each character of the hexadecimal number. Record the decimal equivalents of the characters. The sum of these numbers is the converted number. Hexadecimal number F4D is used as an example.

Hex. Char.	Column	Decimal Equiv.
F	3	3,840
4	2	64
D	1	13
		3,917 = F4D

To convert a number without using the table:

(1) Assign the units decimal equivalent to each hexadecimal character.

(2) Starting with the decimal equivalent of the most-significant character, multiply by 16, add the decimal equivalent of the next most-significant character to the result and again multiply by 16.

(3) Repeat this process until the last character is added.

Hexadecimal number F4D is again used as the example.

```
Hex. Char.   Units Dec. Equiv.
   F              15              15
                                 X16
                                 240
   4               4              +4
                                 244
                                 X16
                               3,904
   D              13             +13
                               3,917 = F4D
```

DECIMAL TO HEXADECIMAL

To convert decimal to hexadecimal using the table:

(1) Select the largest decimal number that is equal to or less than the number to be converted. Record the hexadecimal equivalent as the most-significant character of the hexadecimal number.

(2) Subtract the selected number from the number to be converted.

(3) Select the decimal number that is equal to or less than the result obtained from step 2 and record the hexadecimal equivalent as the second most-significant digit.

(4) Continue the process until there is no remainder.

Decimal number 3,917 is used as the example.

```
Decimal Number
from Table
   3,840             3,917
                    -3,840 → F4D
     64                 77
                       -64
     13                 13
                       -13
```

Conversion without using the table is accomplished by successively dividing by 16 and collecting the remainders in reverse order as shown below.

```
      244
  16 |3917
      32
       71
       64
        77
        64
        13 ─────────┐
       15           │
  16 |244           │
      16            │
       84           │
       80           │
        4 ───────┐  │
        0        ↓  ↓
  16 |15 ───────→ F4D
```

APPENDIX C

HEXADECIMAL/DECIMAL FRACTIONS

Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
.0	.0000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000 0000	.00000	.0000 0000 0000 0000 0000
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625	.00001	.0000 0095 3674 3164 0625
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250	.00002	.0000 0190 7348 6328 1250
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875	.00003	.0000 0286 1022 9492 1875
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500	.00004	.0000 0381 4697 2656 2500
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125	.00005	.0000 0476 8371 5820 3125
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750	.00006	.0000 0572 2045 8984 3750
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375	.00007	.0000 0667 5720 2148 4375
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000	.00008	.0000 0762 9394 5312 5000
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625	.00009	.0000 0858 3068 8476 5625
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250	.0000A	.0000 0953 6743 1640 6250
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875	.0000B	.0000 1049 0417 4804 6875
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500	.0000C	.0000 1144 4091 7968 7500
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125	.0000D	.0000 1239 7766 1132 8125
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750	.0000E	.0000 1335 1440 4296 8750
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375	.0000F	.0000 1430 5114 7460 9375
	1		2		3		4		5

FRACTIONAL HEXADECIMAL TO DECIMAL

When using the table, fractional hexadecimal to decimal conversion is accomplished in the same manner as for integer conversion. Hexadecimal .F4D is converted as shown below:

Hex. Char.	Column	Decimal Equiv.
.F	1	.9375
.04	2	.0156 2500
.00D	3	.0031 7382 8125
		.9562 9882 8125 = .F4D

Conversion without using the table is accomplished as follows:

.F4D = .956298828125

$$.F4D = \frac{F4D_{16}}{16^3} = \frac{3917}{4096} = .956298828125$$

FRACTIONAL DECIMAL TO HEXADECIMAL

Fractional decimal to hexadecimal conversion is accomplished in the same manner as for integer conversion when using the table. Decimal .9563 is converted as shown below.

```
 .9563
-.9375                    = .F
 .0188  0000
-.0156  2500              = .04
 .0031  7500  0000
-.0031  7382  8125        = .00D
 .0000  0117  1875          .F4D
```

Conversion without using the table is accomplished by multiplying successively by 16 and collecting the integers from the products.

```
         .9563
           X16
       15.3008
           X16
        4.8128
           X16
       13.0048
.F4D
```

APPENDIX D

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX E

TABLE OF POWERS OF SIXTEEN

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15