intel®

APPLICATION NOTE

AP-106

September 1980

Multiprogramming with the iAPX 88 and iAPX 86 Microsystems

George Alexy
Microcomputer Applications

AFN-10550A

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Multiprogramming with the iAPX88 and iAPX86 Microsystems

Contents

INTRODUCTION

An engineer, faced with the assignment of developing a multiprogrammed microprocessor based system, need no longer be concerned about the enormity of the task. The new technology and architecture available in today's microprocessors are directed at supporting the basic requirements of these systems. In particular, Intel's iAPX 86 and iAPX 88 microprocessors and support chips will handle this task very efficiently. The purpose of this Application Note is to provide a description of various system requirements for multi-programming and to show how to use the capabilities of the iAPX 86 and iAPX 88 to meet those requirements. As a demonstration of the applicability of the iAPX 86 and iAPX 88 architecture, a multiuser system with Tiny BASIC was created around the iAPX 88. More about this system will be discussed later. For additional information on the iAPX 86 and iAPX 88, the reader is referred to the 8086 Family Users Manual.

WHAT IS MULTIPROGRAMMING?

In any software group, around any desk, you can hear the "buzz" words: multiprogramming, multitasking, multiprocessing, multi-, multi-, multi. A single phrase which will cover all of those buzz words is concurrent processing, which is simply the ability of a system to process more than one function at a time. Multiprogramming, then, is a form of concurrent processing featuring the ability of the system to allow more than one user to access the system's resources at the same apparent time. This does not mean that more than one program is being executed simultaneously; that would require more than one processor and would be multiprocessing. Multiprogramming implies that a processor's time and resources are being divided in such a way that more than one program is executing in the system. If the system is executing much faster than the real time requirements of any program, it appears to an outside observer that all programs are executing simultaneously. What is actually occurring in the system is that the microprocessor, after an initial start-up sequence, will execute one of the programs and after some type of interrupt or polling routine, will process another. (Figure 1). Note this differs from a batch processing environment in which each program runs to completion before the next program starts execution. The interrupt or polling routine will save the state of the machine for the current program and then determine who is going to get service next. It will then restore the previously saved state of the next program to receive service and begin processing this program. After this user has received service for some period of time, the sequence is repeated with a different user. During each context switch, pointers and other required information relevant to the current user are saved and the system proceeds to identify the next user. If there is only one user on the system, that program may get virtually all of the processor's time and resources.

For our design example, a terminal based environment, the input and output are being performed by the operator at human speed, which is extremely slow relative to the speed of the microprocessor. Most of the processor's time in a single-user system is spent waiting for the operator to enter the required information, or for an output device to display the information being sent by the processor. The ratio of effective computer time usage to computer wait time can be very small. Multiprogramming takes advantage of this relatively large amount of wait time by using it to execute a request from one of the other concurrent users. Of course, as the number of users on the system increases, the response time (i.e., the amount of time it takes for the computer to respond to a specific request from an operator) will become longer and longer until it reaches some "unacceptable" limit. In order to maximize the number of users which may acceptably use the system concurrently, the operating system may be "tailored" to a particular type of application.

The operating system is the "master program" which keeps track of what the system is doing and what it needs to do next. It will handle all of the input and output functions such as disk read and write routines, terminal input and output, etc. The operating system also controls the use of system resources, (i.e. allocation of memory to a user) as well as housekeeping associated with switching from one user to another. For example, as a user requests access to a specific program, the requested program and/or data is loaded into the user's work area. The operating system not only loads the requested program, but also may monitor the program's use as dictated by the operator.

Imagine a small accounting system which provides limited service to two or three users. In addition, it has a programmer who maintains the existing programs as well as writing new ones. At any given time during the day, there could be up to four users on the system at the same time, each doing a different task. It is the responsibility of the operating system to ensure that each user's programs and data get loaded and that each user gets the needed service without interfering with the needs of the other users. The intent of this note is to discuss the capabilities of the iAPX 86 and iAPX 88 which support this type of concurrent processing.

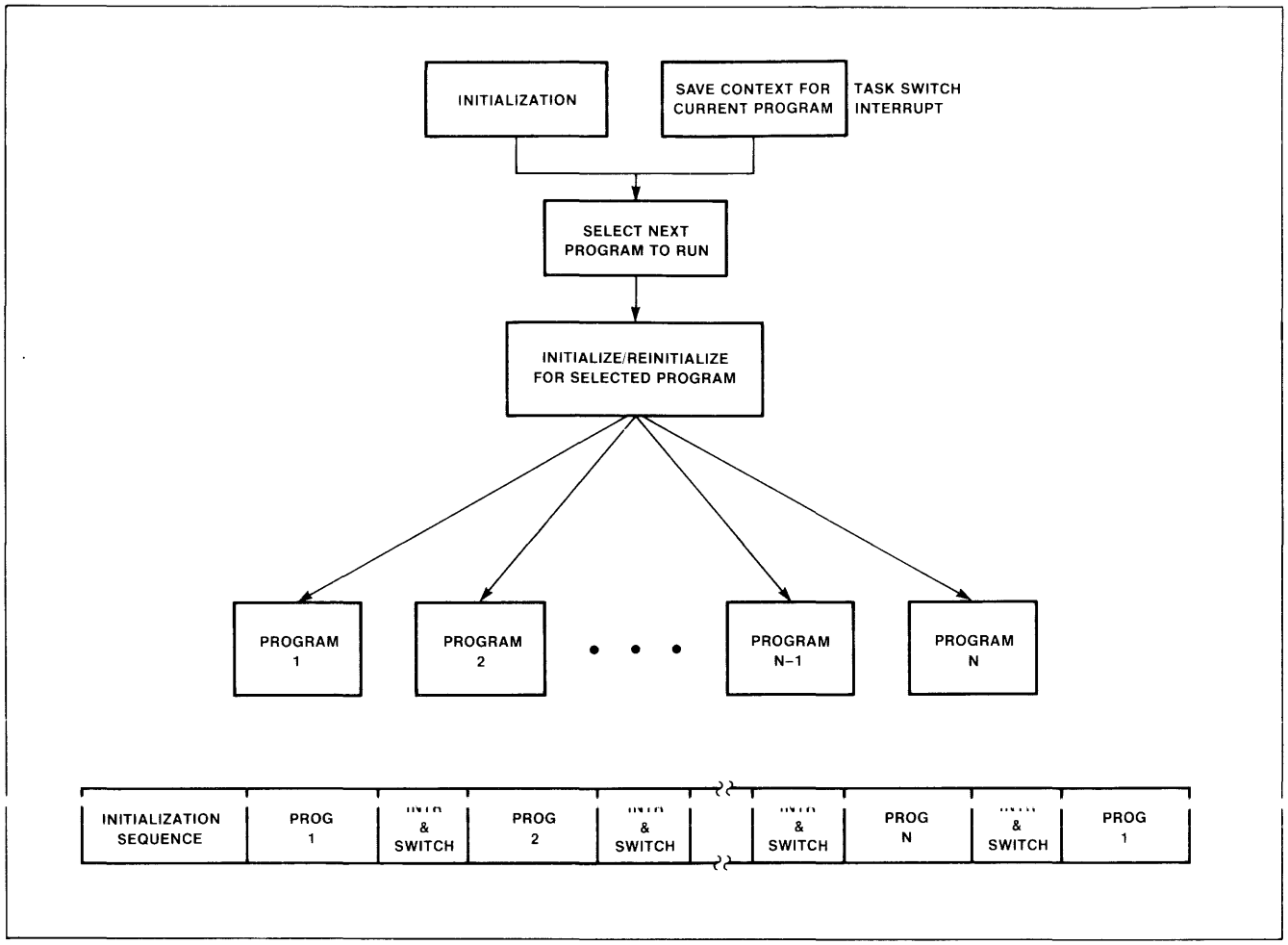

Figure 1. Simple Multiprogrammed System

GENERAL SYSTEM REQUIREMENTS

For a system to provide programming services to more than one user (program) at a time, it needs some method of partitioning the system resources among the users and some method of controlling access to those resources. This requires, from a simplistic view, an operating system which will monitor the user's requests for resources, control allocation of resources and switch the machine state from one user to another.

The primary resources to be shared in the system are the CPU and system memory. To maximize the efficiency and throughput of the system, it is desirable to allow multiple programs and associated data to reside in memory concurrently and switch from one user to another by minimal reloading of the system context. Optimally then, the system will support a simple mechanism for not only allocating areas of memory to each program but also guaranteeing the programs will not violate each others address spaces. This requires that the mechanism for specifiying the currently active (accessible) areas of memory be decoupled (separated) from the general CPU based resources available to the programmer. Strongly associated with this is the requirement for the system to support position independent programs, i.e. programs which will operate correctly regardless of where they are loaded into memory. If the programs are not position independent, then specific programs must reside in specific areas of memory forcing the system to either save and reload memory during task switches or not allow a user to run until the appropriate areas of memory are available.

Another desirable attribute of the system is the ability to support reentrant programs. This capability allows a single copy of a program to be shared by more than one user. This reduces the amount of memory required to support each user by eliminating the need for separate copies for each user or preventing one user from waiting until the program is available for his use. This concept is particularly applicable to system library routines available to all users in the system.

The system also needs to support a mechanism for allocating system resources which cannot be shared (such as peripherals or non-reentrant programs) among the various users. For example, if the system has more users on the system than printers available to print reports, the system needs some method of allocating the printers as the users request their service assuming that each printer is allocated to no more than one user at a time.

In order to share the system, we need a method of determining when it is time to start processing another user's program. This is usually provided by an external interrupt which is input through the hardware interrupt structure of the microprocessor. The external event technique prevents one of the users from monopolizing all of the processor's time, keeping the other users from getting service. An alternate method would be to allow each user to execute until service from the operating system (usually for I/O) is required.

When the system is interrupted, it needs some method of determining who gets to use the system next thus allowing the amount of time the processor has available to execute programs to be allocated among the various users on the system.

APX 86 AND iAPX 88 ARCHITECTURAL FEATURES

Intel's iAPX 86 and iAPX 88 directly support many of the features required for multiprogrammed applications. The architectural capabilities of the family stem from the register structure, large memory address space, almost unlimited interrupts, a powerful segmentation scheme and addressing structures which support reentrant and relocatable programs. The segmentation scheme allows the processor to provide capabilities such as program sharing, dynamically relocatable code and functionally partitioning memory among multiple users with a degree of protection implemented directly by the CPU.

Segmentation

Segmentation is the partitioning of a program and its data into specific elements called segments. Basically, segments are assigned to logical (and often variable length) elements (i.e. code or data) and should not be confused with the term "page" which is typically associated with a fixed length area of memory. Using segmentation, a programmer may assign modules of his program to the segments, and his data structures to additional segments (Figure 2). All references to a specific logical element (either program or data) are made relative to the appropriate segment. Assuming that the user's program does not specifically modify the segment registers, the operating system may place each segment anywhere in available memory and the program will still function properly. This concept of position independence becomes very important in multiuser systems where the number of users and the number of different programs in the system varies over time. As a user begins to initiate tasks in the system, the operating system will allocate specific areas in memory for the required segments. As more and more users enter the system they also are given system resources with which to execute their programs. Using segmentation, it doesn't matter what areas in memory the segments are assigned because the programs reference the data and programs relative to the associated segments. If the programmer breaks the data and programs into different segments, they may be located separately anywhere in memory. This method provides efficient utilization of the system memory resources since memory is only allocated for the specific segment size. The concept of segmentation also provides a degree of isolation between users in the system if all program and data references are restricted to their own segments. The ability to separate code and data into separate segments supports the concept of shared programs by allowing each invocation of a single program to reference only data and temporary variables relative to the data and stack segments of the current active user. Segmentation is a powerful concept which, up to this time, has been available only in larger computers such as minis and mainframes. Note that in other architectures,

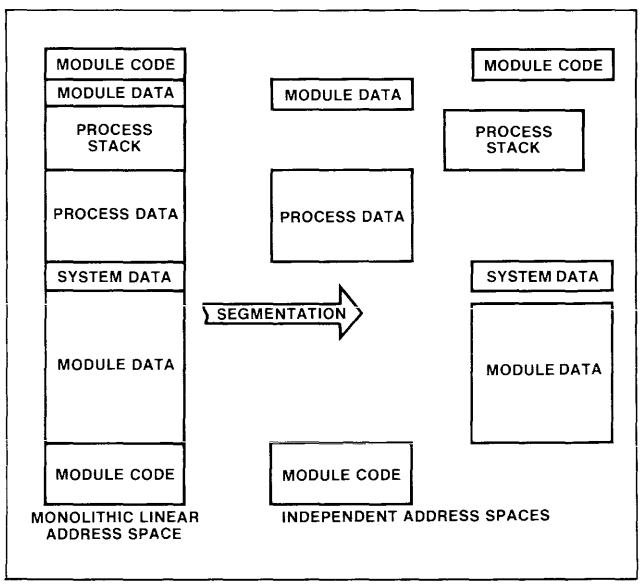

Figure 2. Segmented Address Space Partitioning

the notion of position independent data is supported through the use of based addressing where a base register resource is used to identify the location of data and all references are made relative to the base register. The primary drawback with this approach is its restriction on the flexibility with which based addressing can be used, particularly when dealing with data structures and arrays. The concept of segmentation allows defining the position (within memory) of the structure without consuming the base register resource (Figure 3). This concept is supported directly by the CPU architecture of the iAPX 86 and iAPX 88.

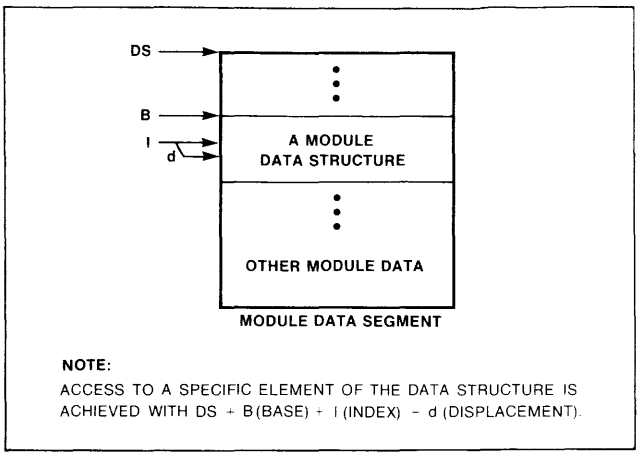

Figure 3. Four Component Addressing Usage

Segment Registers and Usage

The iAPX 86 and iAPX 88 concepts of segmentation define four types of segments: Code Segments, Data Segments, Stack Segments, and Extra Segments. Each is associated with a CPU based segment register which points to the currently active (being accessed) segment of each type.

Each segment register is 16 bits and defines the starting address of the segment within the iAPX 86's or iAPX 88's one megabyte address space. The segments can start on any 16 byte boundary and may vary from 16 to 64K bytes in length with length resolution on 16 byte boundaries. This implementation optimizes memory usage by allowing the segment size to be defined within 15 bytes of the size of the logical element the segment contains. The segment size assigned to each element is independent of other segment definitions and allows supporting anywhere from sixty-four thousand 16 byte segments to sixteen 64K byte segments within the one megabyte address space.

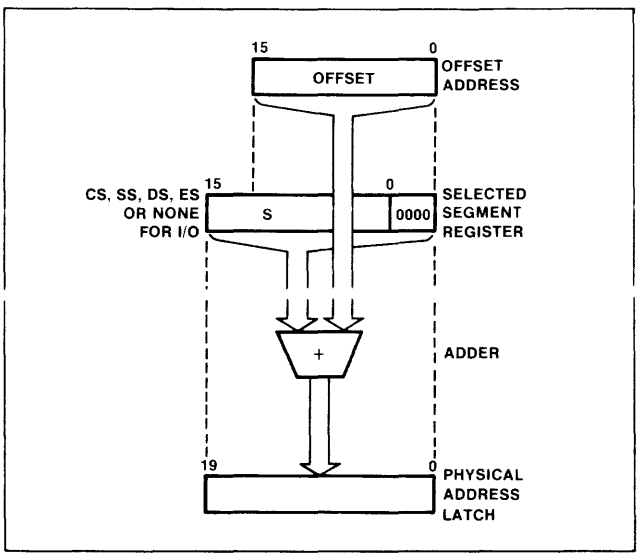

Figure 4. Address Generation with Segment Registers

To reference memory, the value in the segment register selected for a specific memory reference is automatically multiplied by 16 by appending a suffix of four binary zeros, and is added to an offset address specified for each access (Figure 4). The result is a 20-bit address which may be used to access anywhere within the one megabyte of directly addressable memory. The data operand offset address calculation is determined by the selected or implied addressing mode given in the instruction. The available modes allow for greater flexibility in the manipulation of data structures than possible with other types of architectures (ref. 8086 Family Users Manual).

Since the offset address is 16 bits in length, programs that are: (a) less than 64K bytes of code (the maximum segment size); (b) do not change the segment register values and, (c) reference data contained within single data and extra segments (up to 128K bytes of static data), are considered directly relocatable and isolatable from other programs in the system. Considerations for programs which extend beyond a single code segment or two data segments are shown in Appendix 1. The system, therefore, has four segment registers: one to define the data, one to define the stack, one to define the code, and an extra segment which can be used to specify another data segment or global (shared) system data. Since the iAPX 86 and iAPX 88 support based stack segment relative addressing for access to operands on the stack, the stack is typically used for dynamic allocation of workspace and storage of temporary variables in addition to parameter passing during procedure invocation. The technique of dynamic allocation of memory for temporary space reduces the need for static data space resulting in more efficient use of memory. Using the stack for dynamic data provides support for reentrant procedures as demonstrated in Figure 5.

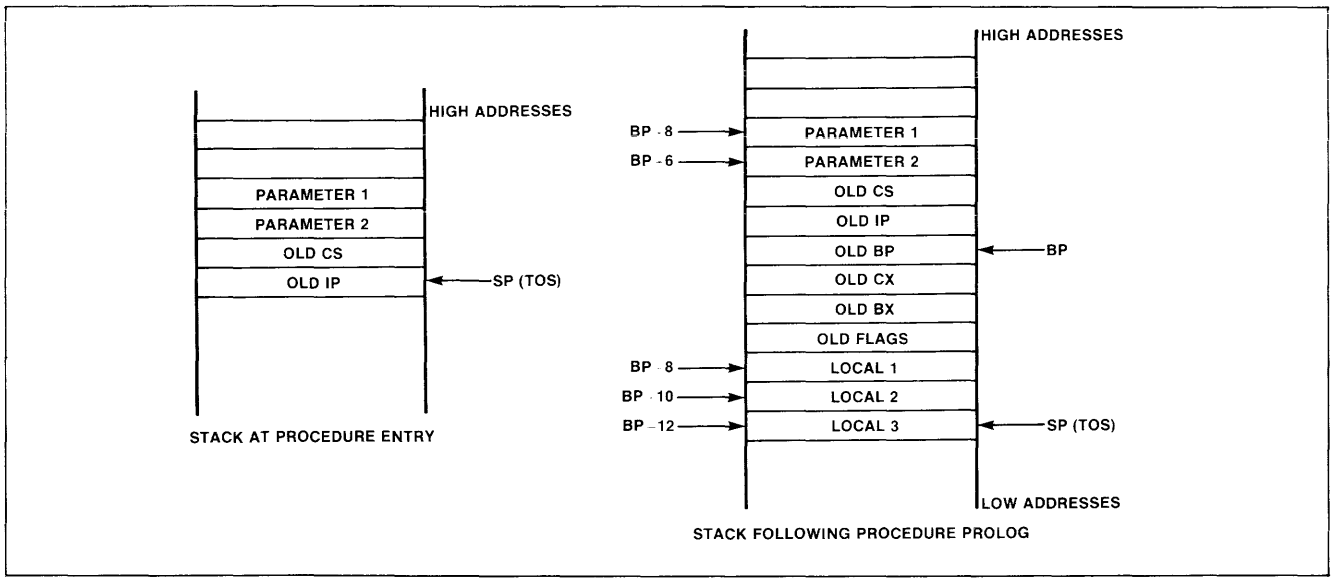

Figure 5a. Stack Image for Reentrant Procedure

```
EXAMPLE     PROC  FAR       ;MUST BE ACTIVATED BY
                            ;    INTERSEGMENT CALL

;PROCEDURE PROLOG
            PUSH  BP        ;SAVE BP
            MOV   BP, SP    ;ESTABLISH BASE POINTER
            PUSH  CX        ;SAVE CALLER'S
            PUSH  BX        ;    REGISTERS
            PUSHF           ;    AND FLAGS
            SUB   SP, 6     ;ALLOCATE 3 WORDS LOCAL STORAGE
            ;END OF PROLOG
;PROCEDURE BODY
            MOV   CX, [BP+8] ;GET ELEMENT COUNT
            MOV   BX, [BP+6] ;GET OFFSET OF 1ST ELEMENT
            ;PROCEDURE CODE GOES HERE
            ;FIRST PARAMETER CAN BE ADDRESSED:
            ;  [BX]
            ;LOCAL STORAGE CAN BE ADDRESSED:
            ;    [BP-8], [BP-10], [BP-12]
            ;END OF PROCEDURE BODY
;PROCEDURE EPILOG
            ADD   SP, 6     ;DE-ALLOCATE LOCAL STORAGE
            POPF            ;RESTORE CALLER'S
            POP   BX        ;  REGISTERS
            POP   CX        ;  AND
            POP   BP        ;  FLAGS
            ;END OF EPILOG
;PROCEDURE RETURN
            RET   4         ;DISCARD 2 PARAMETERS

;ASSUME ARRAY_1 IS INITIALIZED
;
;CALL "EXAMPLE", PASSING ARRAY_1, THAT IS, THE NUMBER OF
;ELEMENTS
; IN THE ARRAY, AND THE LOCATION OF THE FIRST ELEMENT.
            MOV   AX,SIZE ARRAY_1
            PUSH  AX
            MOV   AX,OFFSET ARRAY_1
            PUSH  AX
            CALL  EXAMPLE
```

Figure 5b. Reentrant Procedure "EXAMPLE" Using the Stack

Using the segment registers, the system can provide each user not only a separate memory space for his data, but also an individual program area. By changing the segment registers, the processor may define which program is executing with which data. By specifying different segment values, users may execute different programs. In this way, one user can access a BASIC interpreter while another is using a FORTRAN compiler and a third is using something different. This becomes very useful in large systems where there will be more than one user executing concurrently. The basic system context which must be saved and restored to transition from one user to another consists only of the CPU registers and does not require reinitialization of memory or off chip address translation devices.

With regard to multiuser environments, segmentation not only provides the ability to partition the memory space, but also allows the system to change the areas in memory being accessed by a specific program by changing the segment registers before entering the program. Most iAPX 86 and iAPX 88 microprocessor instructions (all except those which specifically modify the segment registers or pass control of the processor to an area outside the current code segment) access memory relative to the current values of the segment registers. When the operating system wants to move the user to a different area, it only needs to move the program or data and change the appropriate segment register values (Figure 6). The program will continue execution unaffected by the relocation. Using this method, as new users enter the system, their programs are loaded into available memory areas by the operating system. When a program is invoked, the operating system will set the segment registers based on where the program and data are located. This also allows the operating system to reformat the allocation of memory and minimize memory fragmentation as users enter and leave the system. This provides the system with position-independent programs because the users are not dependent on executing at a specific memory location.

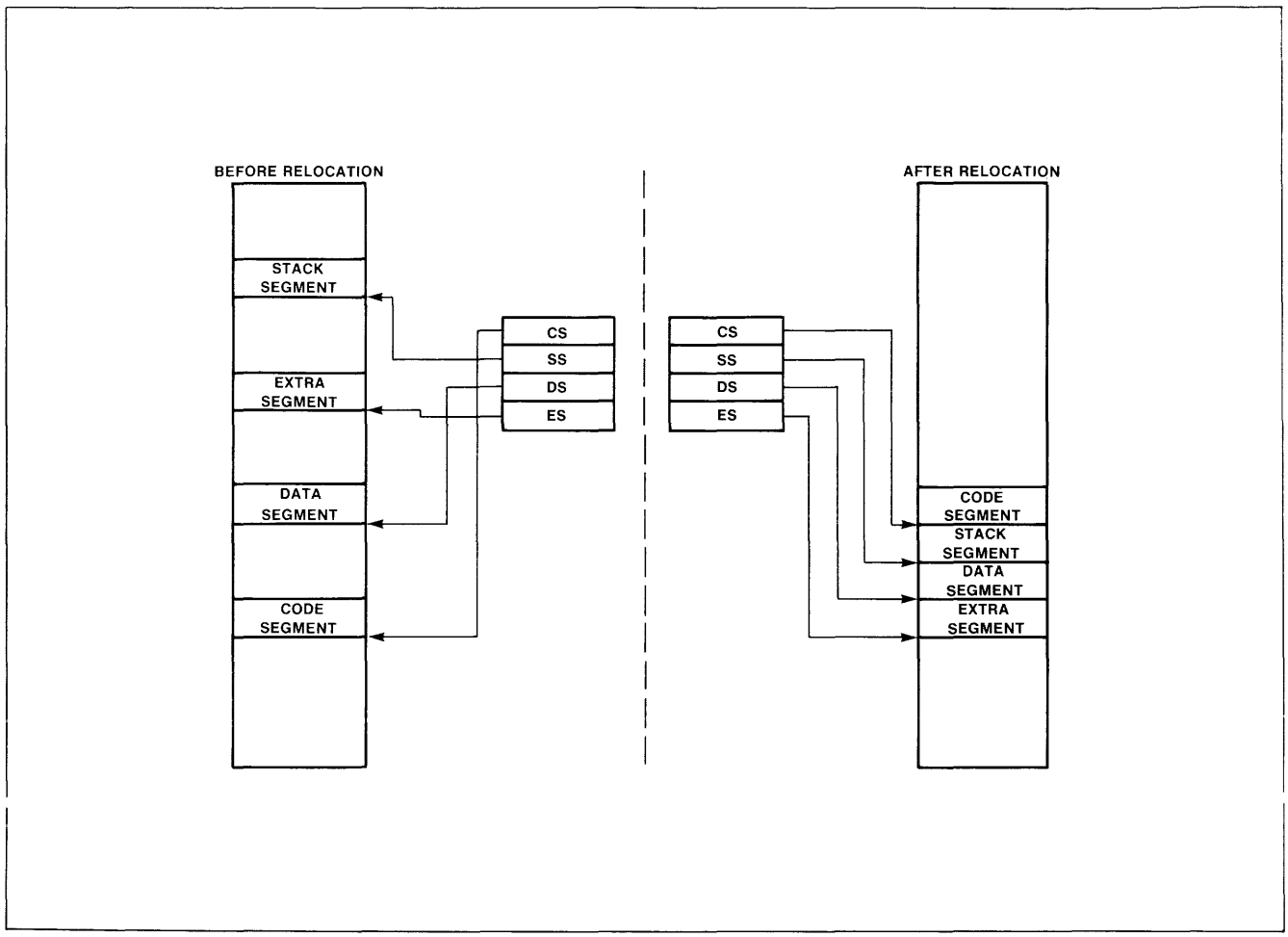

Figure 6. Dynamic Code Relocation

The ability to relocate programs anywhere in memory also facilitates the use of multiple 'master' programs by the system. The programs which are requested may be loaded into any user's workspace and not be dependent upon being in a specific location to be useable. In this manner, each user may have his own copy of a program while each program simultaneously resides in any number of locations.

Alternatively, by setting the code segments of two users to the same value, they will both access a single copy of the same program. Of course, in order for the program to execute correctly, it must be reentrant. This implies that as a user is accessing the program, the program cannot be self-modifying and the data the program accesses (including temporary variables defined within the program) must be uniquely defined and private for each user accessing the program. The reentrant program is independent of the users accessing it and is accessible to any number of users, independent of the user's status.

Using this concept, a program executed by more than one user such as a BASIC interpreter or a COBOL compiler may be written so that it always looks for specific pointers or values in locations relative to the active user's data, stack or extra segment registers. When a user wants to invoke this program, the operating system sets the segment registers to point to the user's workspace before initiating execution for this user. When the shared program accesses the memory, it would access the current user's memory.

The ability to have one program process several users' data by merely changing the appropriate segment register (usually the Data and extra segments) simplifies the implementation of multi-user programs. In order for several users to access the same program, it does not need to be duplicated in each user's workspace (Figure 7). As long as the program is reentrant, once the program has been loaded into system memory, any user may access the program by setting the Code Segment and instruction pointer to the entry point of that program.

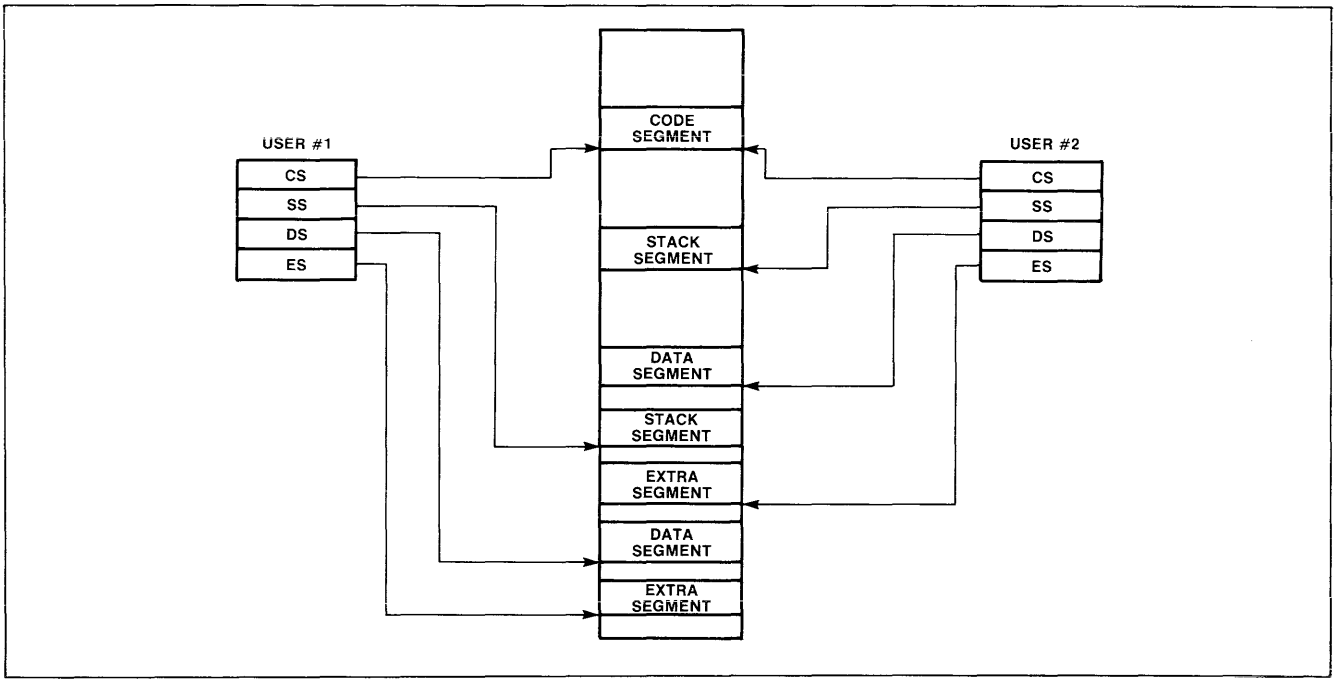

Figure 7. Single Copy of a Program Shared by Multiple Users

Semaphores

The system, in addition to the features described earlier, needs some method of allocating resources (memory and external devices or peripherals) when the number of users on the system may exceed the number of system resources available. The system needs the ability to reserve a device or area in memory for a given user and to prevent other users from accessing the resource until the first user releases. It wouldn't work too well if after the first half of a balance sheet was printed, the system started to print a program listing. The purpose of semaphores is to control access to resources, providing a mechanism for a single user to gain access to the resource and allowing other users to determine easily whether the resource is available or not. There are several instructions available to the iAPX 86 and iAPX 88 user which will help him program semaphores into his system. They will be discussed in the following examples. One method of implementing a semaphore is to reserve a byte for each resource for which access must be controlled. The byte will contain information as to whether a device is being used by someone or whether it is available for use. When a user wishes to reserve a specific resource he checks to see if it is available and if it is, a value specifying that the resource is busy is loaded into its semaphore. The user may then access or gain control of the resource. When someone else wishes to see if the resource is available, all he need do is test the semaphore value and see if the resource is available. If not available (busy), the user must wait until the resource becomes available. When the user controlling the resource is done, he must reset the busy indication in the semaphore to allow others to gain control of the resource.

Even in systems that use semaphores to allocate system resources, problems may still arise if the operating system is not programmed properly. For example, imagine a system which has one printer. In order to specify if the printer is being used or not, the system uses a byte in memory. When the system allocates the printer to a user, it places a 1 in the byte to tell other users that the printer is already allocated. Another user may see if the printer is busy by reading the byte and seeing if it is a 1. If not, the printer is available for use.

Now assume that User 2 and User 4 are both setting up reports to run on the printer. User 2 is currently being processed. He tests the byte which will tell him if the printer is busy or not and finds that the printer is not busy. He now knows that the printer is available and is about to reserve the printer for his report. But, before he can store the appropriate value in the byte telling the other users that the printer is busy, he gets interrupted and User 4 begins operations.

When User 4 checks the semaphore to see if there is a report already running on the printer by seeing if the semaphore has a value of 1, it finds that the printer is still available. User 2 did not change the value of the semaphore yet; he was interrupted too soon. So, User

4 sets the semaphore to a 1 to tell the other users that the printer is busy and begins to print his report on the printer. Eventually the processor returns to User 2. Now, User 2 has already checked the semaphore and "knows" that the printer is available so he now loads the semaphore with a 1. (This was also done by User 4 when he took the printer but was not rechecked by User 2 when he returned.) What follows is two different reports being merged as both users send information to the printer.

Within a multiprogrammed single processor environment, there are several methods of dealing with this problem. One is to disable the interrupts before User 2 tries to find an available printer. If this were done then he wouldn't be interrupted until he had had a chance to specify that the printer was in use by loading the semaphore with a 1. After this was done, the interrupts would then be enabled. However, in many systems it is undesirable to allow a user to modify the interrupt system and this method may not be acceptable. Another method, probably easier for the programmer to implement, is to use the iAPX 86/ iAPX 88 XCHG (Exchange) instruction when programming this situation. The instruction exchanges the value in one of the operands with the other operand (Figure 8). For User 2 to check the status of

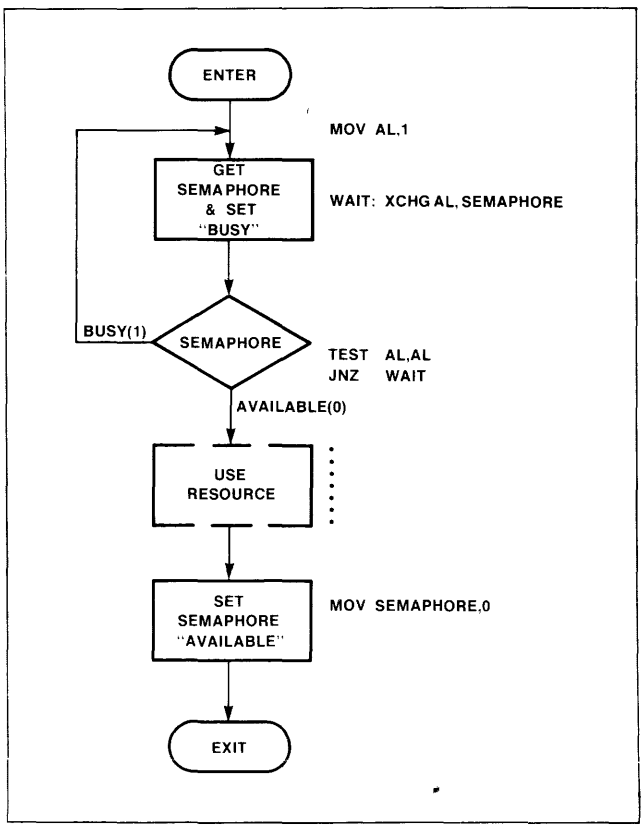

Figure 8. Semaphore Test and Set for Single Processor—Multiprogrammed Environment

the printer, he would first load a register with a 1, specifying that the printer is busy. This value is then exchanged with the value in the semaphore. If anyone interrupts User 2 after the exchange, they will find a 1 in the semaphore and they will know that the printer is busy. User 2, on the other hand, may examine the byte loaded into the register from the semaphore to see if the printer is available. If the printer was already assigned, User 2 would have exchanged a 1 with a 1, the semaphore would have been left the same and User 2, after checking the value in the register, would know that it was busy. If User 2 finds a 0 in the register, he knows that the printer is not busy and that it is now assigned to him. He has already indicated that the printer is being used so he does not need to reload the semaphore with a 1. Since the exchange operation is a single uninterruptable operation of reading and setting the semaphore, the problem in the previous example is avoided.

In a multiprocessor environment, an additional mechanism is required which will prohibit one processor from accessing the data bus while another processor is in the middle of the exchange operation. If both processors attempt to access the semaphore at the same time, one or the other may get erroneous results. While a multiuser system with only one processor will be interrupted only on an instruction boundary, in a multiprocessor environment, access to the bus is commonly shared on bus cycle boundaries. To allow the programmer or system designer to prevent other processors from gaining control of the bus during the exchange with semaphore operation, the iAPX 86 and iAPX 88 have a bus lock feature. LOCK is a special one-byte instruction prefix which will cause the processor to emit a bus-lock signal for the duration of the instruction that the LOCK precedes. The prefix may be placed before any processor instruction. Using the prefix bus arbitration circuitry will lock out all other processors for the duration of the instruction. The programmer may then protect any critical data areas from outside modification until the processor has had a chance to complete the operation being performed.

THE 9-CHIP MULTIPROGRAMMING SYSTEM

To show the performance and architectural capabilities of the iAPX 86 and iAPX 88 in a multiprogramming environment, a small system was developed around the iAPX 88 microprocessor. The iAPX 88 system is based on an 8-bit bus CPU with the full programmer visable architecture of the 16 bit bus iAPX 86. This includes full object code compatibility between the iAPX 86 and iAPX 88 as well as 16 bit

data types, 1 megabyte address space and addressing modes. The advantage of the iAPX 88 in small systems such as the one we will discuss, is compatibility with the multiplexed bus memory and peripheral devices of the 8085 Family. As a result, this system uses nine chips to provide all of the system clock signals, I/O ports, interrupt signals, user workspace, and operating system. In this case, the multiprogramming system is a multiple user Tiny BASIC Interpreter. Written originally to demonstrate the 8085, the code was converted to execute on the iAPX 88 using CONV86. CONV86 is available as part of the general set of iAPX 86/iAPX 88 software development tools and converts 8080/8085 code to iAPX 86/iAPX 88. The iAPX 88 Tiny BASIC Interpreter is a reentrant program to allow multiple users.

System Overview

The software structure for the system is shown in Figure 9 and consists of a simple O.S., Tiny Basic and work space for each user. The O.S. handles terminal I/O and time sharing of Tiny Basic between users. Each user is allocated a separate stack, temporary variable work space, I/O line buffer and BASIC program area. The physical address space for each user is defined by the contents of the segment registers.

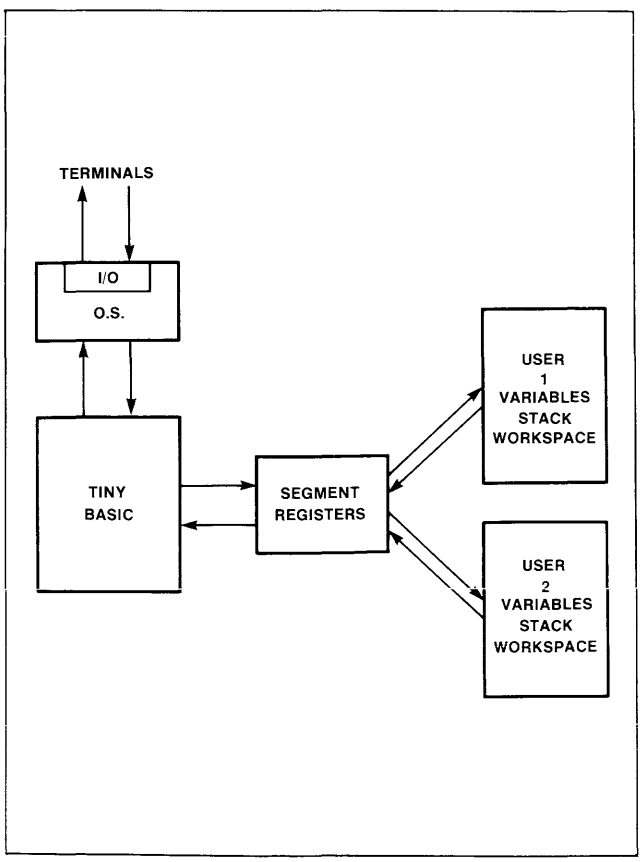

Figure 9. Multiuser System Structure

The O.S. is effectively an interrupt handler for timer interrupts used for I/O. During each interrupt, the current users machine state is saved on his stack, the stack and data segments are switched to the next user and a return is executed. The return restores the machine state for the next user from his stack and returns to Tiny Basic execution for the next user. Between the saving of machine status for one user and restoring status for the next user, the O.S. performs pending terminal I/O.

The Tiny Basic program transfers control to the O.S. to perform I/O. This suspends Tiny Basic execution for the current user until the O.S. completes the requested I/O. Upon completion of the I/O, the O.S. will return control to Tiny Basic for this user. Tiny Basic makes no distinction between users and references all data relative to the current data and stack segment register values. Both users have the same values for the code segment register values. Both users have the same values for the code segment with the individual IP values depending on each users execution sequence.

Hardware Description

The total system consists of: an iAPX 88 microprocessor; 8284 Clock Generator to provide the clock signals to the processor and a programmable timer; 8155-2 to provide 256 bytes of RAM, 20 I/O lines and the programmable timer used to generate interrupts; two 8755A-2's which provide additional I/O lines as well as the EPROM where the BASIC Interpreter programs and the operating systems are stored; and two 8185's which provide 2K-bytes of RAM for the users' workspace. Communications between the user terminals and the processor board is accomplished with a 75189 and 75188 for level conversion from TTL to EIA standards.

The 8284, driven by a crystal, provides the clock signal for the iAPX 88 and the programmable timer on the 8155-2. The suggested crystal, 15 MHz, will provide a 5 MHz signal to the microprocessor and a 2.5 MHz signal to the timer/counter. The 8284 also provides the system reset to the processor and the other chips.

The 8155-2 provides the system with 256 words of RAM and is connected directly to the iAPX 88 multiplexed bus. The system uses the programmable timer on the 8155-2 to generate the baud rate interrupt signals to the processor. The timer output is connected to the non-maskable interrupt pin on the processor. When an interrupt is generated, the processor enters the interrupt routine which performs the

proper input and output sequences for communication with the terminals and switches from one user to the other for execution of Tiny BASIC. The timer count is set during the initialization sequence to count to a specified value. When it reaches this value, an interrupt signal is sent and the timer begins counting again. Since this timer is set in software, the baud rate is software programmable. In addition to the timer, the 8155-2 provides two programmable 8-bit I/O ports and one 6-bit I/O port.

The 8755A-2 is a 2K-byte EPROM. Two of these chips were used to provide enough space for the operating system and the Tiny BASIC Interpreter. In the final system, approximately 60% of the total EPROM memory was used; the remaining space is available to expand the capabilities of Tiny BASIC or to increase the number of users the system can handle. Like the 8155-2, the 8755A-2 is directly compatible with the iAPX 88's multiplexed bus. In addition to the EPROM there are two 8-bit programmable I/O ports on the chip. Each line in the 8-bit port is individually programmable to be either an input or an output line. Two I/O lines on one of the 8755A-2's provide the I/O for one of the terminals. Two lines on the 8155-2 provide the communications to the other terminal.

The other chip used in the system is the Intel 8185. This chip is a 1K-byte RAM device. Two were designed into the system to allow each user 1K-bytes of memory for program and workspace while generating and executing BASIC programs. This chip, like the others, is compatible with the iAPX 88 multiplexed bus. Additional space to facilitate larger programs or increasing the number of users can be accommodated by increasing the number of 8185's in the system.

All of the chips used in this system are directly compatible with the 5 MHz iAPX 88 system bus. Therefore, no latches or data bus transceivers are needed in the system. Linear select techniques were used to select all devices and eliminate address decode logic. Figure 10 shows a functional diagram of the nine-chip system. To implement multiple users, the TIMER-IN line on the 8155-2 was wired to the PCLK line on the 8284 and, the TIMER-OUT line was wired to the non-maskable interrupt line on the iAPX 88. The timer was set to operate the terminals at 300 baud.

Software Description

In addition to the BASIC Interpreter, three programs were needed to provide the multiuser capabilities. The first was an initialization routine which is invoked after system reset. The next was the Character-In/Character-Out routine which is used to communicate with the user's terminal. The last was the Interrupt routine. This routine is called each time the 8155-2 sends an interrupt to the processor.

The two EPROM chips are selected between addresses FF000H and FFFFFH, each having 2K-bytes of memory. These were placed at upper memory since the system accesses addresses FFFF0H from system rest. When this address (FFFF0H) is placed on the address bus, the chip corresponding to FF800H to FFFFFH is activated. The two 8185 RAM chips are selected when addresses between 1000H and 17FFH are placed on the address bus. The first addresses (1000H to 13FFH) are used to hold the data and programs for User 1. The second addresses are used to hold the data and programs for the second user. The 256-bytes of RAM in the 8155-2 are used by the operating system for information it requires that is not directly associated with either of the users. The 8155-2 RAM is selected by addresses OH to FFH and contains the interrupt vector table. The vector table contains the addresses of the routines the system will execute if the user attempts to divide by zero or when the 8155-2 sends an interrupt signal. These addresses are loaded into the interrupt vector table during the execution of the initialization routine following reset. An address map of system memory is shown in Figure 11.

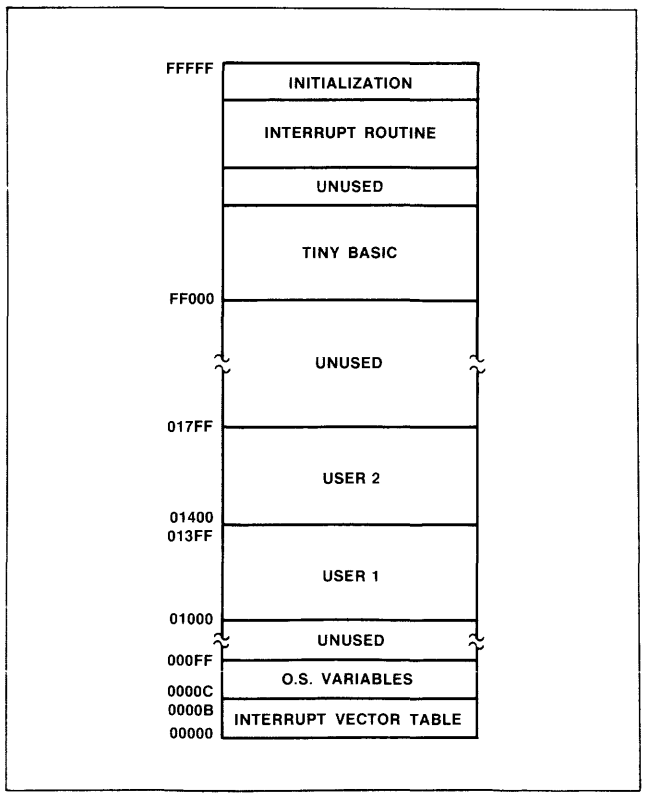

Figure 11. Memory Map

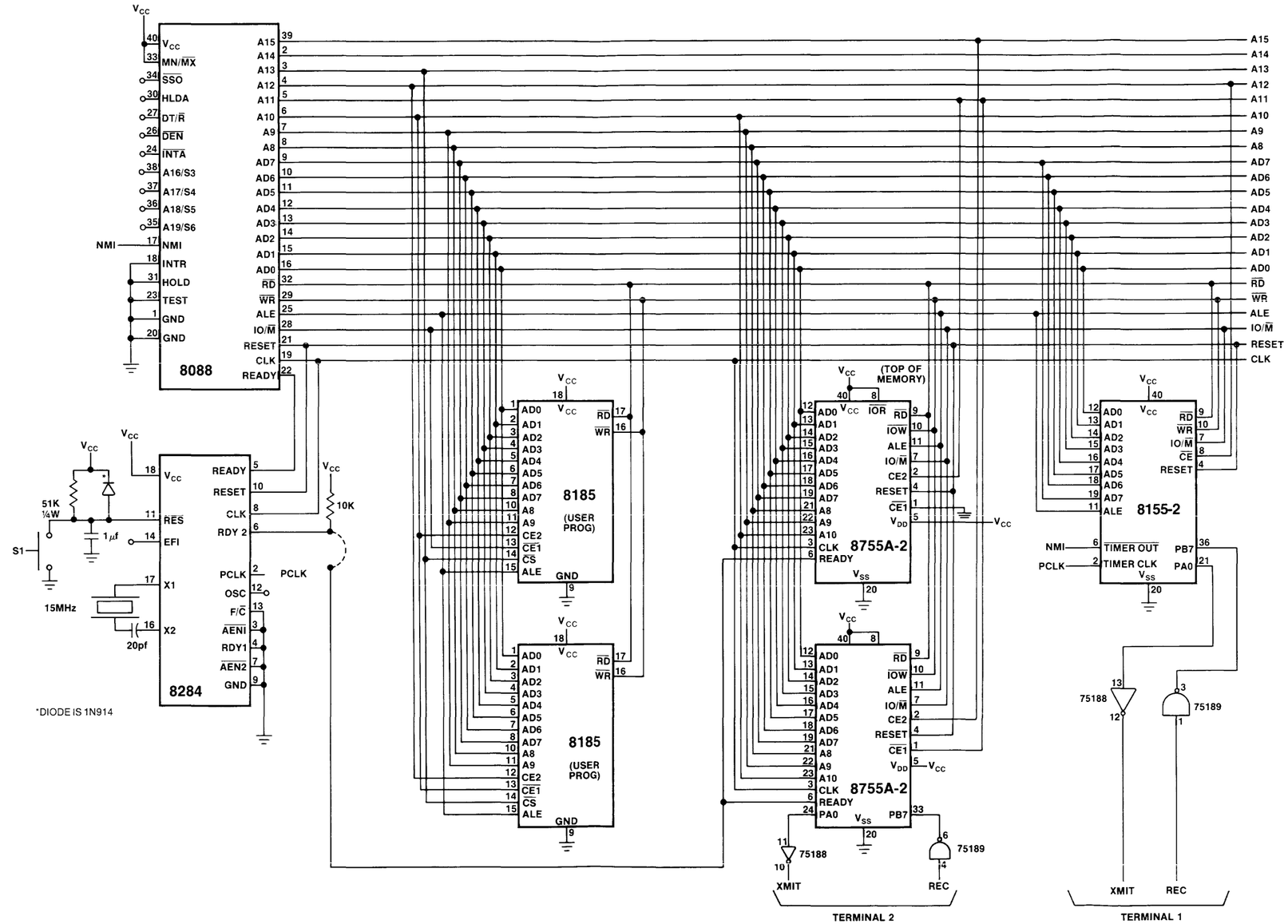

Figure 10. Schematic of 9 Chip 8088 Multiuser Tiny Basic System

First, the initialization routine (Figure 12) sets up the data segment registers so that they point to the data and program area reserved for User 1. The routine then resets all data and program pointers which are reserved for User 1. These pointers and data areas are used by the BASIC Interpreter in keeping track of the processing being done by User 1. This includes pointers which address the beginning and the end of the BASIC program, the data variables, and the information required to perform the I/O with the terminal. After the data and pointers have been initialized for User 1, the routine moves this information into the data and pointer area reserved for User 2, thus initializing the system for the second user. Usage of each user memory space is given in Figure 13.

After the routine has initialized the pointers for both users, it sets up the interrupt vector table. The first entry points to the error routine which will be called if the user attempts to divide by zero. The second entry is the location of the TIMER-OUT routine which will be invoked when the timer sends an interrupt to the NMI input. After the interrupt vectors are initialized, the iAPX 88 initializes the I/O ports which are used to communicate with the two terminals. Since each bit of each port may be programmed as either input or output, they must be defined by the system before they may be used. After the lines are programmed for their defined purpose (one input and one output line for each user), the system outputs a high signal to each of the output ports, sending a STOP bit. This STOP bit will remain valid until the BASIC Interpreter is ready to send a message or data to the user's terminal.

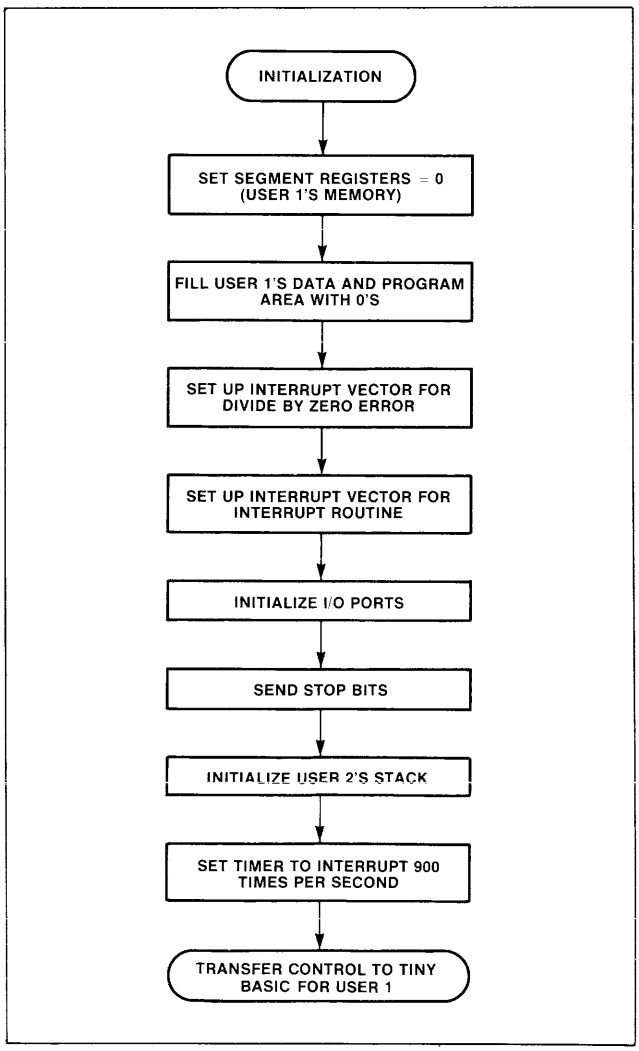

Figure 12. Initialization Routine

The system software consists of an operating system that handles terminal I/O and sharing CPU processing intervals between the users. There is a single shared user program (Tiny BASIC), and stack and workspaces for each user. To initialize both users with Tiny BASIC, the reset initialization procedure initializes User 2's stack to point to the starting address of Tiny BASIC, sets up the stack extra and data segment register to point to User 1's stack and workspace, enables the 8155 timer for baud rate generation and transfer control to Tiny BASIC for User 1. The CPU then continues to execute Tiny BASIC on behalf of User 1 until a timer interrupt occurs. The interrupt transfers control to the operating system which performs the proper I/O, switches to the next user's segments and returns to the next user. Note the state of the machine for User 1 is saved on his stack. Since User 2's stack was initialized to point to the start of Tiny BASIC, the system now begins executing Tiny BASIC on behalf of User 2. For each interrupt, the operating system may return to either user by loading the proper segment values and returning to where that user had previously been interrupted.

The programmable timer is initialized so that it will generate 900 interrupts per second. The system samples the incoming data from the terminal three times for each bit coming in, and the system communicates at 300 baud. The system samples each bit three times so that the accuracy of the input may be improved. When the system jumps to START (beginning of Tiny BASIC) for each user, the BASIC Interpreter will print "OK" on the terminal and wait for the user to begin entering data on the terminal. For the BASIC to print "OK" on the screen, and to monitor the input from the user's terminal, the program uses the Character-In/Character-Out routine in Tiny BASIC.

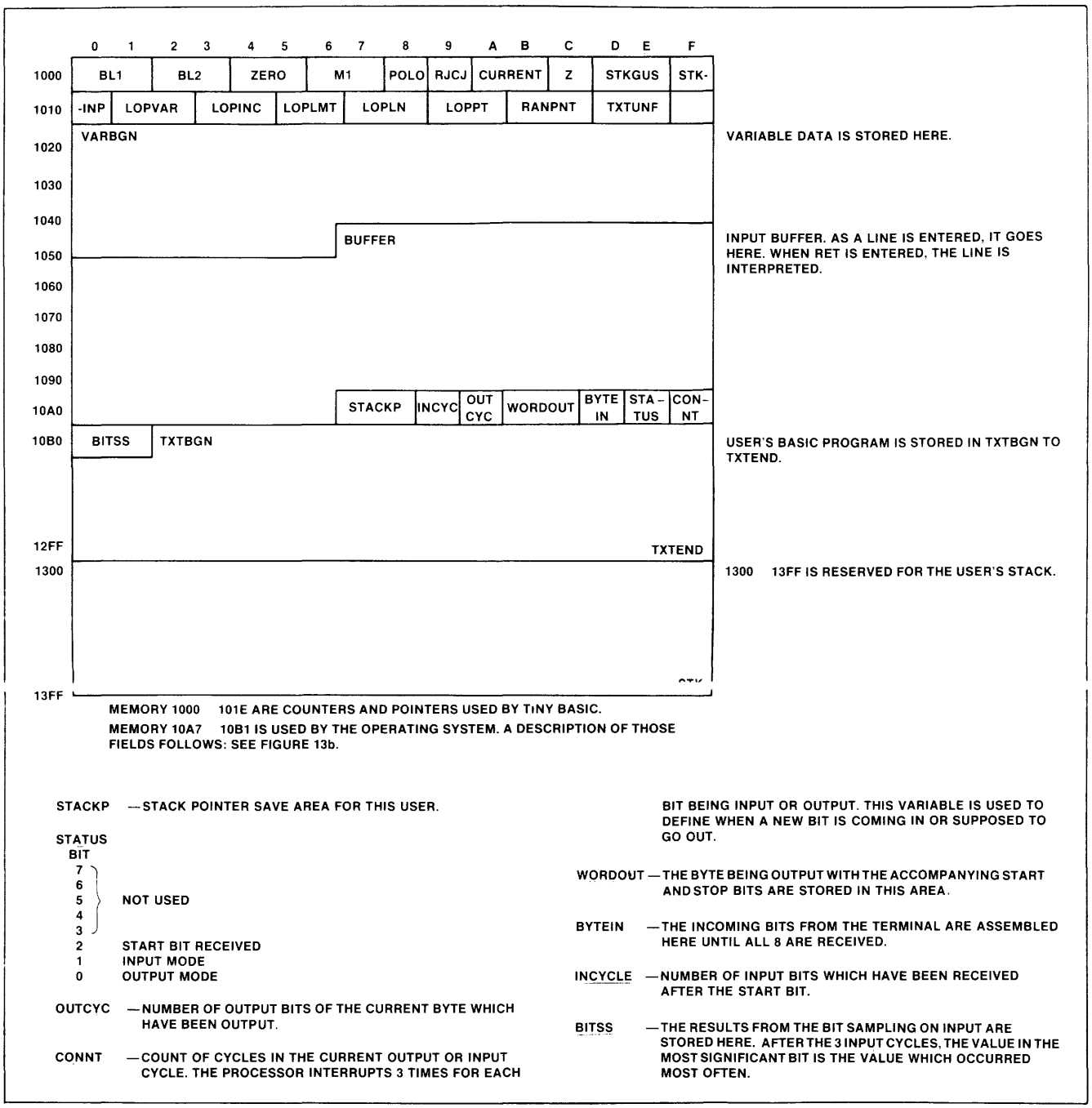

Figure 13. Definition of each User's Memory Space

The Character-In/Character-Out routines are very simple. Figure 14a is the sequence for a user to enter and exit I/O while figure 14b is a more detailed flowchart for the CI and CO routines. The Character-In routine (CI) sets the count of bits received (INCYCLE) to zero and sets the status (STATUS) for this user to a 2 so that the interrupt routine will know that this user's program is waiting for something to be input on the user's terminal. It pushes onto the user's stack all of the registers and segment registers as if the user had been interrupted, then transfers control to a wait loop in the operating system. It will loop like this until the timer issues an interrupt. When in the I/O routines for a specific user, the system is waiting for a user to input data or waiting for the proper timing to output bits to the user's terminal, and will not process that user's program. This effectively suspends the user's execution until the user requested I/O is complete. When the system has completed the I/O (read a character from the terminal or output one to the terminal), the user's stack for the completed operation is modified so that the operating

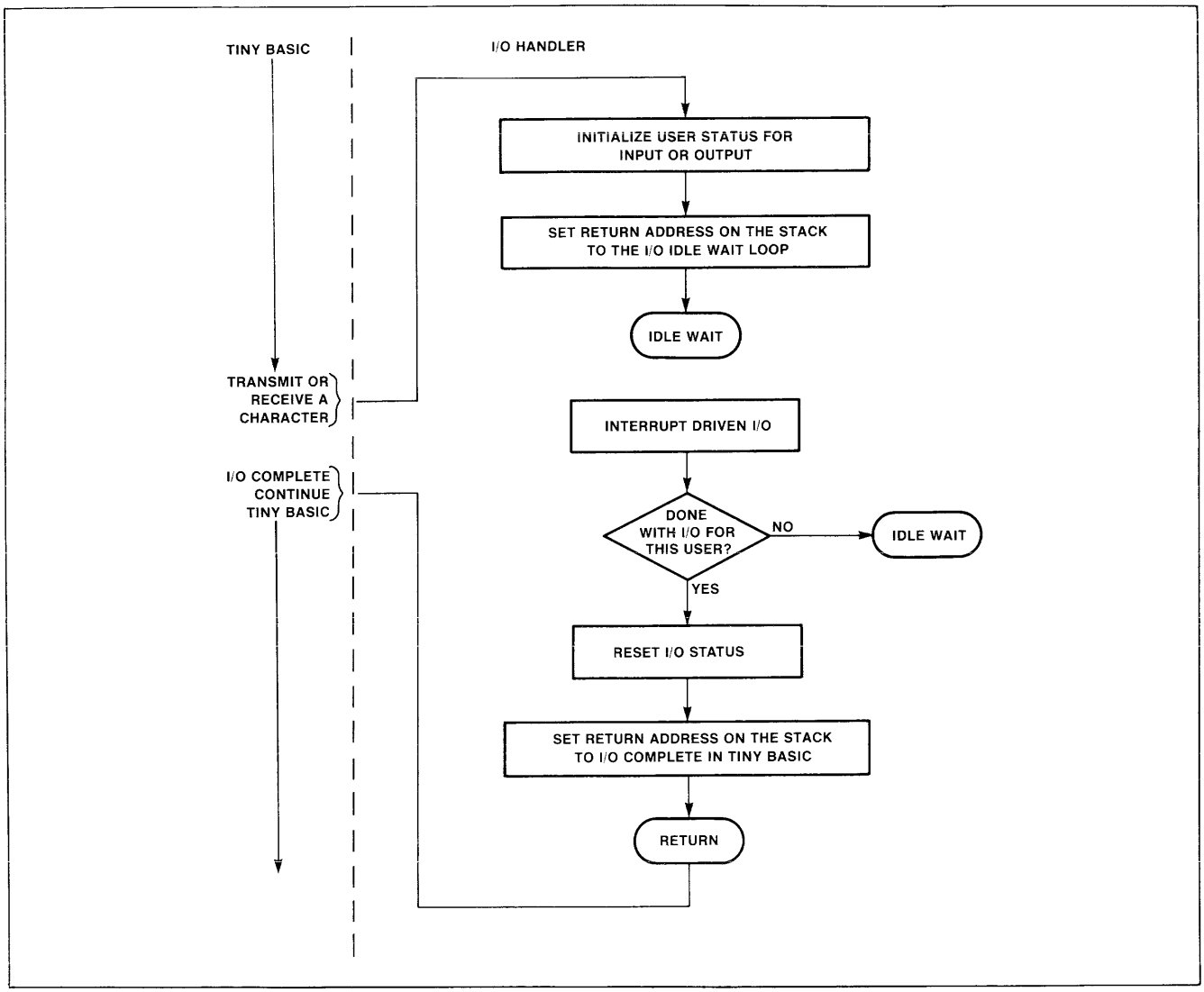

Figure 14a. Sequence for a User to Enter and Exit I/O

system will return to the calling program. While one user is doing I/O, the other is able to process normally, and the processor will give virtually all of its time to the other user.

After the current user is in the I/O mode, the processor attempts to identify who should receive service next. If either of the users is not in an I/O mode, he will receive the control of the processor. If both of the users are in an I/O mode, the system will loop until an interrupt occurs.

The OUTPUT routine works in the same manner as the INPUT routine. The character to be output to the user's terminal is formatted with the appropriate START and STOP bits and is stored in the user's memory (WORDOUT). The status byte (STATUS) is set to indicate to the interrupt routine that this user is going to be outputting a byte to the terminal. After the user's memory is initialized to perform the output to the terminal, his stack is set, as in the CI case, to loop at IORTI until the whole byte has been sent to the terminal. The actual input and output of information to and from the terminal is accomplished by the Interrupt routine.

The interrupt routine is called each time the timer/counter on the 8155-2 reaches the count assigned in the Initialization routine. This count may be modified so that the system can run at any desired baud rate. To determine what the count should be, divide the number of clock cycles (2,500,000) by three times the desired baud rate.

Since the software must handle each bit in the serial I/O stream and samples at 3x the baud rate to eliminate synchronization problems, 300 baud was chosen

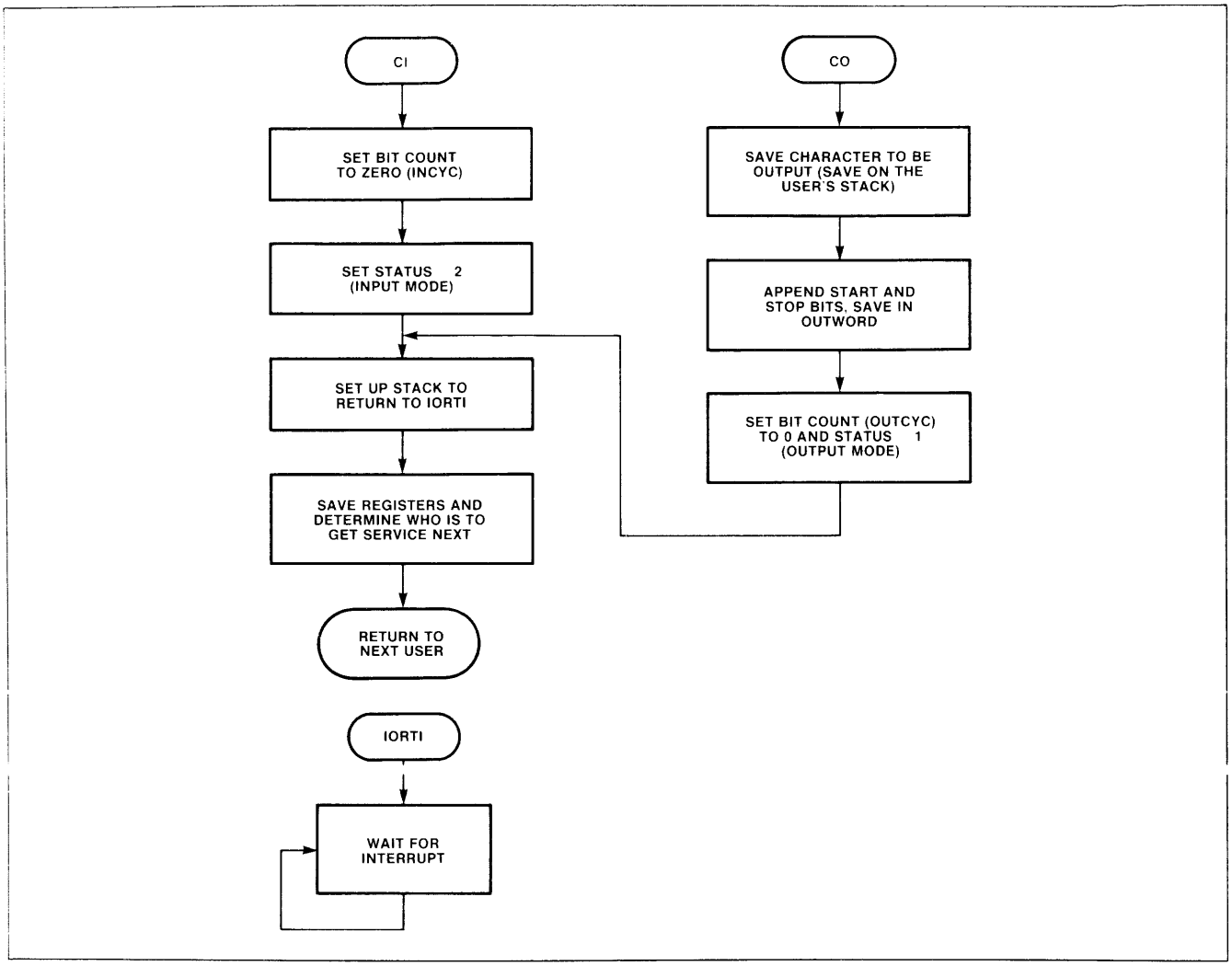

Figure 14b. Character In/Character Out Routines

to provide reasonable I/O speeds and accuracy while avoiding impacting performance for individual users in the multi-user environment.

When the user is interrupted, the interrupt issued by the timer causes the processor to automatically save the flags and the return address onto the interrupted user's stack. This enables the system to tell where to return when it is that user's turn to be processed. The interrupt routine then saves all of the registers for the user. When the system returns to the user, it will have the same values in all of the registers so that the program can continue as if it had never been interrupted.

After all of the registers have been saved, the actual processing of the interrupt can begin. The interrupt routine has two functions. The first is to perform the proper inputs and outputs for the BASIC Interpreter. The second is to identify who is to get service the next time that the processor begins to execute a user's program. Note that if neither user is performing I/O, the interrupt service routine simply switches users (Figure 15).

In performing the inputs and outputs for the BASIC interpreter, it is imperative that the signals sent to the terminals and the checks for incoming data occur at the same point in time for each interrupt cycle. Doing this will ensure the accuracy of the data as it is read from the terminal and will ensure that the output is at the proper baud rate. To get the I/O to occur at the same time in the interrupt cycle, the processor must always perform the same steps. If the system is required to first check the status of the users and then perform the appropriate operation depending on the status found, the processor would be going through different steps, depending upon whether the user was in an INPUT mode, an OUTPUT mode, or a regular processing mode. This in turn would cause the I/O to come at different points in the interrupt routine. To allow the system to perform the I/O at the same time

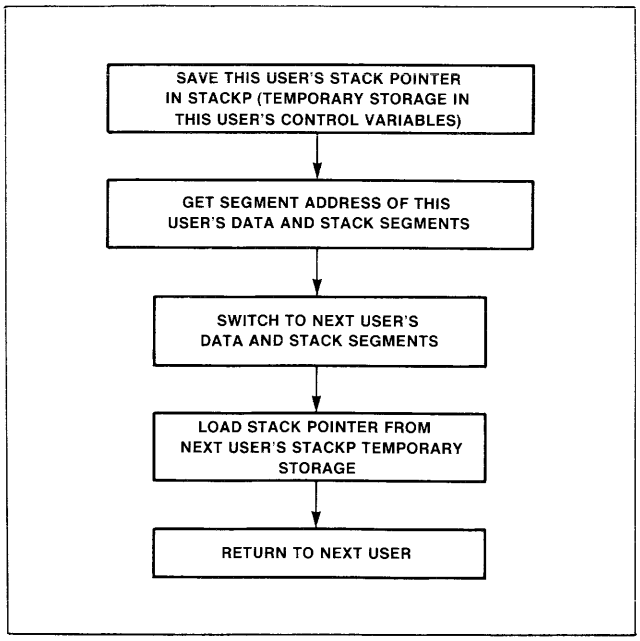

Figure 15. Sequence to Switch Between Users

in each interrupt cycle, the processor always performs input as soon as all of the registers of the current user have been saved. It doesn't check to see if the user is in an INPUT mode; it doesn't really matter. It then takes the data received from the input and saves it for future processing. Later on, after performing the time critical I/O functions, the processor will look at the status (STATUS) for each user to see if each user is in an INPUT mode. If the user is in an input mode, the processor will use the data received. If the user is not in an input mode, the data will be ignored.

After it has performed the inputs and saved the data, the processor performs the two outputs, one for each user. Again, it does not check to see if the terminal is in OUTPUT mode; it always outputs something to the terminal. If a user is not in the OUTPUT mode, the processor will send a STOP bit to the terminal. The system uses each user's status byte (STATUS) to generate the STOP bit if the user is not in OUTPUT mode or to leave the data the same if the user is in OUTPUT mode. In either case, the steps followed to output a bit to the terminal are always the same to keep the timing signals constant.

It may be interesting to see how the processor outputs either a stop bit or the next bit of data. This processing is done in the OUTWORD subroutine (Figure 16). The first thing the routine does is check to see if it is time to send another bit. Since the processor is interrupting three times for each bit being sent, the output should only be changed every third interrupt. CONNT is a variable used to count the bit cycle. Each time the count in CONNT reaches 3 it is changed back to a 0. Each time the OUTWORD routine sees a 0 in CONNT it outputs the next bit to the terminal.

The first three lines of OUTWORD are checking this count to see if it is time to send another bit. If not, it jumps around the code where the bit is generated and outputs to the terminal. If it is time, the processor loads the output character to the accumulator. The next bit of the character is transferred from the register to the terminal, low order bit first, one bit every CONNT interval.

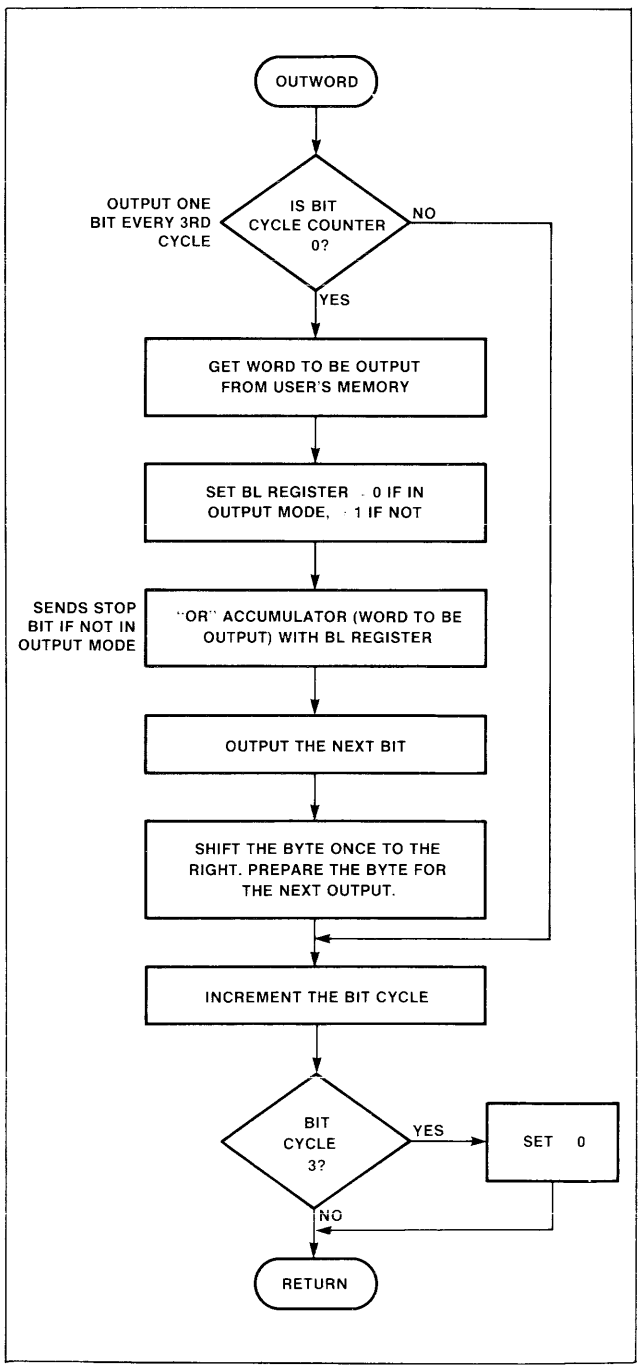

Figure 16. Output Data Processing

After the appropriate value has been output to the port, the value of the byte being output is shifted to the right one bit, preparing the field for the next time a bit is to be sent to the terminal.

Once the proper outputs have been made to the terminals, the processor begins checking the status of each user and taking appropriate action. If the user is in OUTPUT mode, CONNT is checked to see if a new bit was just output to the terminal. If it was, the count of bits sent to the terminal (OUTCYC) is incremented. For each character the BASIC Interpreter wants to send to the terminal, 10 bits must be output. This includes the 8 bits of data in the byte plus a START bit and at least one STOP bit. When the routine has output the full 10 bits, the user status code is reset to 0 so that the processor will know that the user is no longer in the OUTPUT mode. Next the user's stack is modified so that it will return to the calling program rather than the wait loop.

If the user is not in an OUTPUT mode the system checks to see if he is in an INPUT mode and results in entering the INBYTE routine (Figure 17). If he is in an INPUT mode the system checks to see if the user has received a START bit from the terminal. If the user is waiting for a START bit, the input received from the terminal is tested for a START bit. If it is not, the processor ignores the input and continues to wait until a START bit has been received. When a START bit is finally received, the user status (STATUS) is set to indicate the START bit has been received. CONNT is also initialized so that the input will be received and interpreted correctly.

If the user is not waiting for a START bit and is in INPUT mode, the input received from the terminal is valid data. For each interrupt cycle the processor performs an input at the beginning and one more when it determines that it is in an INPUT mode. These two inputs are performed for each of the three cycles, giving six data inputs from which to determine the value of the bit being sent by the terminal.

The variable BITSS is initialized to OOFFH. (It is stored as a 2-byte word). As each of the inputs is received from the terminal, the value in BITSS is shifted to reflect the data received. If the data received is a 1, BITSS is shifted left once. If the data received is a 0, the value in BITSS is shifted right once. After all six inputs have been checked and BITSS has been shifted accordingly, the value which occurred most often will be indicated by the high order byte of BITSS. The newly received bit is OR'd into BYTE IN. After this sequence has occurred eight times, the bit which was entered first will be in bit 0 and the subsequent entries will follow. Once the whole byte has been received, the user's stack is modified so that the return address, is updated to return to the Character-In routine rather than the wait loop. Here, the value assembled in BYTE IN is placed in the accumulator and the system returns to the program where the Character-In routine was called.

After the interrupt routine has checked all of the I/O and has performed the appropriate action concerning the users' modes, the system determines which user is to receive service next. It first looks at User 1. It checks to see if he is in an INPUT or OUTPUT mode. If User 1 is in either, the system will not start User 1 but will automatically begin processing for User 2. (If User 2 is also in an I/O mode, the system will loop until the next interrupt from the timer.) If User 1 is not in an I/O mode, the system checks User 2's status. If User 2 is in an I/O mode, the system will automatically give User 1 service next. If neither of the users are in an I/O mode, the system will return to the user who has waited the longest for service. This is accomplished by examining who was executing when the system was last interrupted, and then setting the segment registers to the other user. In this way, the system is shared between both users. If one of the users must wait while in an I/O routine, then his time allocation is given to the other user until the user waiting in the I/O routine has completed the I/O.

The system changes users by performing an Exclusive OR of 40H with the segment register of the user who was first given service. After the system determines who will be serviced next, it restores the new user's segment registers and then restores the registers and flags which were pushed onto the stack. Thus, the user's status is restored before the interrupt return. The system performs an IRET (Interrupt RETurn) which restores the flags to their original value and returns control to the interrupted program. With all of the registers and flags restored, it appears to the BASIC Interpreter user as if there had never been an interrupt and processing will continue normally.

As long as the BASIC Interpreter references all data relative to the segment registers and does not change the segment registers (is reentrant), the system will handle the two users without difficulty. If the program attempts to change the segment registers, then User 1 may interfere with User 2's data or programs, or vice-versa.

Using this type of operating system, the Tiny BASIC used here could be replaced with any other program which is reentrant. Since all of the users use the same "master" program, there is no need to move one program out when the users change; the system only needs to save the registers of the current user on the

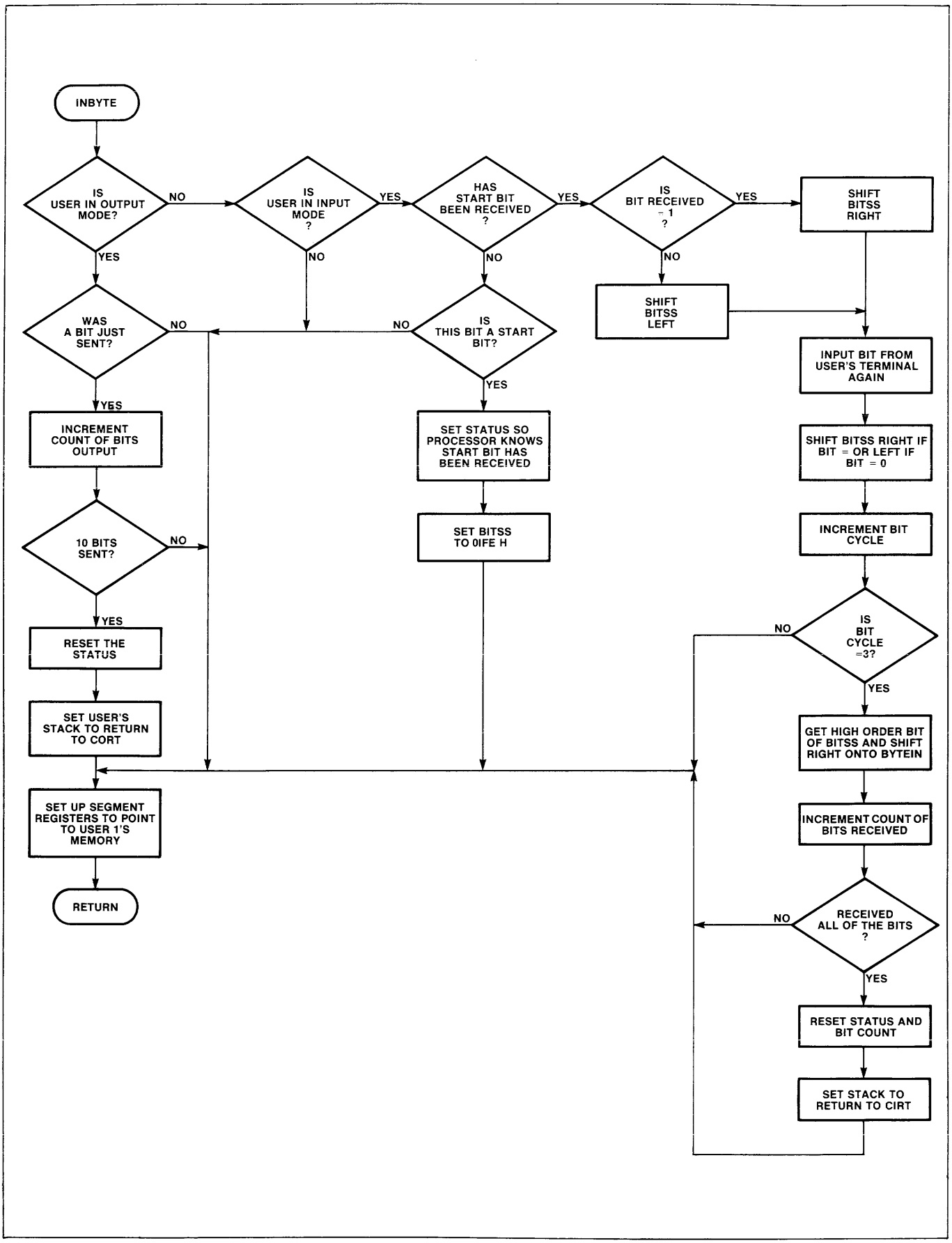

Figure 17. Inbyte Routine to Assemble Input from the Terminal

user's stack before it attempts to go on to the next user. Code for these functions are included in Appendix 2. The complete multiuser Tiny BASIC program is available through INSITE, the Intel Users Software Library.

Adding More Users

With BASIC or any other "master" program set up this way, it is a simple job to change the operating system so that it will support more than two users. The system, as currently written, assumes that each of the users has a 300-baud terminal and a specific amount of memory available for his data and BASIC programs. These are the areas that someone who desires to add more users to the system will have to modify. The Initialization routine, which initializes the stack of User 2 and the I/O ports which communicate with the terminals, will also have to initialize the stack and the I/O ports of any additional terminals. This means that more memory will be required or that the existing memory will need to be subdivided among the number of allowable users. Also, of the remaining 48 I/O lines which are not currently being used, two lines for each additional terminal will be assigned for communications. If the memory boundaries are changed, one of the pointers in BASIC which defines the size of the buffers will have to be modified to reflect the new size of the buffers.

The Initialization routine will, as it does now, go to the "master" program's START for the first user. When the first user is interrupted or begins I/O, the system would need some method of determining whose turn it is to receive service. This process could be handled much like the present routine. After the system finds out who is not in an I/O routine, it will process the user who has been waiting the longest for service by restoring that user's registers, and returning to that user. If all users are in an I/O mode, it will loop until the next interrupt. As soon as the first user finishes with the I/O routine, the system will continue to process that user's program until other users also continue execution. Then the system will switch users as before.

The Interrupt routine, which now performs the I/O for only two users, would need to perform the inputs and outputs for each additional user, and assemble input or disassemble output as they are received or sent. As long as there are remaining I/O lines and enough memory for each of the users, any number of new terminals could theoretically be added to the system if user response time is not a major consideration, and as long as the interrupt routine can complete execution before the next timeout interrupt occurs.

Variant Considerations

With the extra I/O lines, and additional ROM space not used by the BASIC Interpreter or the operating system, there are other features which could be added to the system if desired. A printer could be added to the system with a special output routine to tell the processor that the user's program is trying to output to the printer instead of the terminal. This would allow the user to generate hard-copy reports or program listings. To do this, would require addition of commands to the BASIC Interpreter so that the user who desires to print on the printer can do so in his BASIC program. The BASIC Interpreter would also need to provide a method where the user could check to see if the printer is available. Some type of semaphore is required so that the other users can tell if the printer is in use, and reserve it for their use if it is available.

The Interrupt routine would then be changed so that it would give the proper output signals to the printer, both when in use and when idle. The type of output to be sent depends upon the type of printer and the signals the printer expects. Also, the initialization routine would need to be modified so that it would send the proper initialization signals to the printer when the system is reset.

Since there are multiple lines available on the I/O ports, other types of peripherals could be added to the system as long as there are methods by which the users could request and release the devices and there is enough room in the BASIC Interpreter to add the commands to allow the BASIC programs to access the device.

CONCLUSIONS

Admittedly, we have shown a relatively simple example of a multiprogrammed system with most of the system complexity evolving around totally software driven, simultaneous I/O for both users. It is just this point, however, which is noteworthy. The basic architectural characteristics of the iAPX 86 and iAPX 88 allowed us to simply (almost trivially) implement the reentrant Tiny BASIC interpreter and the operating system primatives for time multiplexing use of the interpreter between two separate users. In a similar vein, the architecture also supports the capabilities of dynamic relocation and controlled access to critical regions which would be required by more sophisticated systems. With the addition of operating system capabilities for memory allocation and management, loader and file I/O a more general purpose system could be developed.

APPENDIX A

CONSIDERATIONS FOR PROGRAMS WITH MULTIPLE CODE AND DATA SEGMENTS

To extend the concepts of relocatability to programs which consist of multiple code and data segments, the iAPX 86 and iAPX 88 support the ability to transfer control indirectly through memory and load data segment addresses from memory based tables. These capabilities may be implemented in various ways depending upon the characteristics of the code generators, load modules and loader. The basics of any implementation are:

1) Transfer of control to all external procedures or labels must be indirect through memory.

2) References to each data segment must be preceded by loading the data segment register from a table containing the location of each segment in memory.

The tables must be constructed by the loader at load time of the program and data, and maintained by the operating system if the segments are relocated. The location of the tables are implementation dependent. If programs are RAM based, the tables may be appended to the code segments and accessed relative to those segments, each segment maintaining its own table of data segments and external code references (Figure A.1).

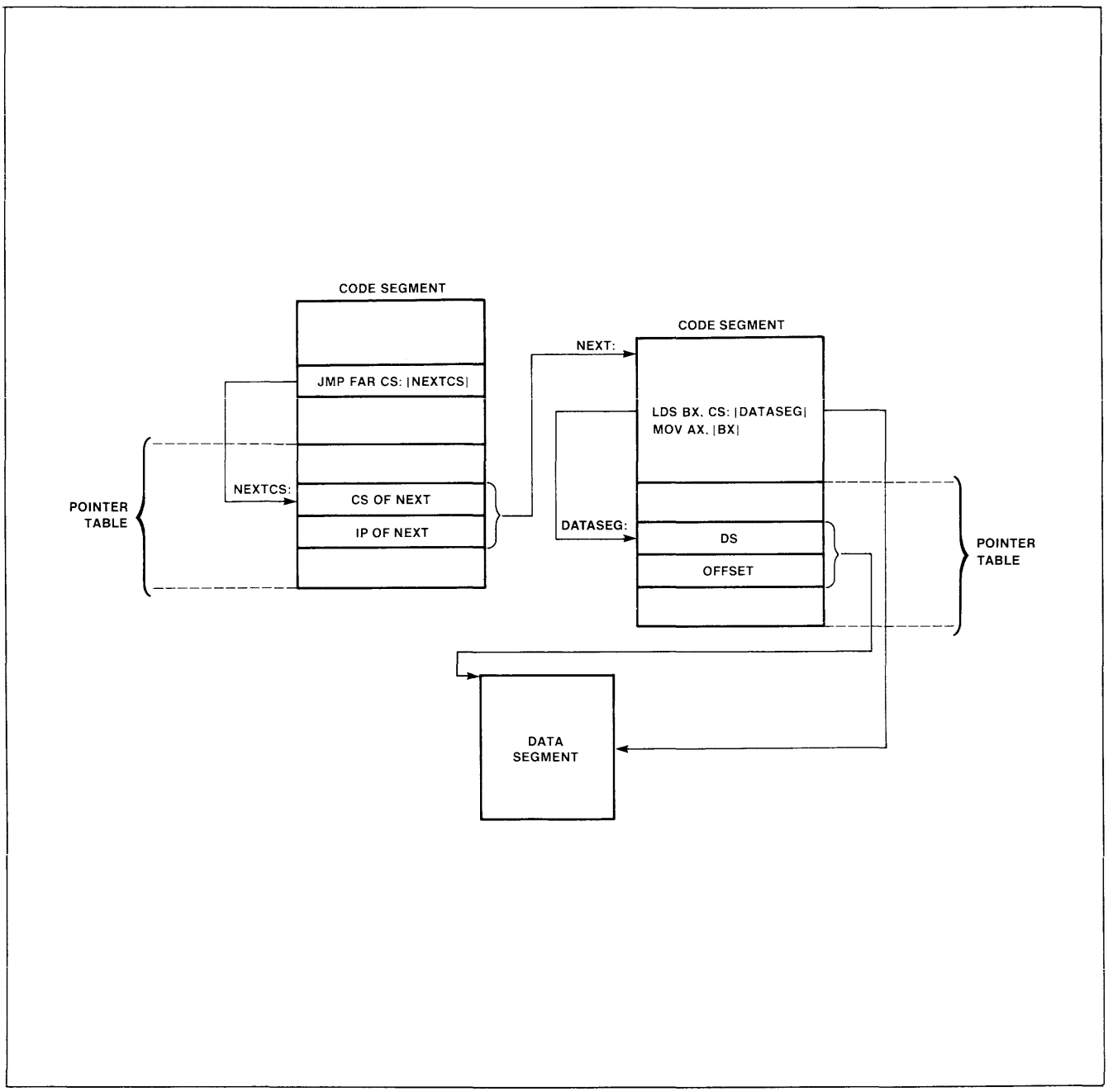

Figure A1. Table of Pointers to other Code and Data Segments associated with each Code Segment

An alternate technique (Figure A.2) would allow a single table to be constructed for use by all code segments and contain the location of all code and data segments. If the programs reserve use of the extra segment for containing this table, the loader and O.S. need maintain only one table rather than one within each code segment. Another benefit of this approach allows the code segments to be in ROM without fixing their location in memory for all possible instances of their use. This is particularly applicable to library routines that will be used in a variety of end applications. A transfer of program control would then require loading the code segment and instruction pointer values from the ES based table. The displacement into the table is specified in the control transfer instruction and therefore, must be specified during linkage or preparation of load modules. Likewise, the location of each data segment must explicitly be loaded with the LDS instruction. The LDS instruction must reference the table in the ES segment and contain the displacement to the appropriate data segment address.

These techniques support programs and data which are relocatable at load time but not necessarily dynamically relocatable (i.e., operation is suspended, code and data are relocated and execution resumes). Since stack based return addresses and pointers are real addresses, dynamic relocation of code or data based on these techniques would require fixing up stack resident segment values, in addition to jump tables, before resuming task execution.

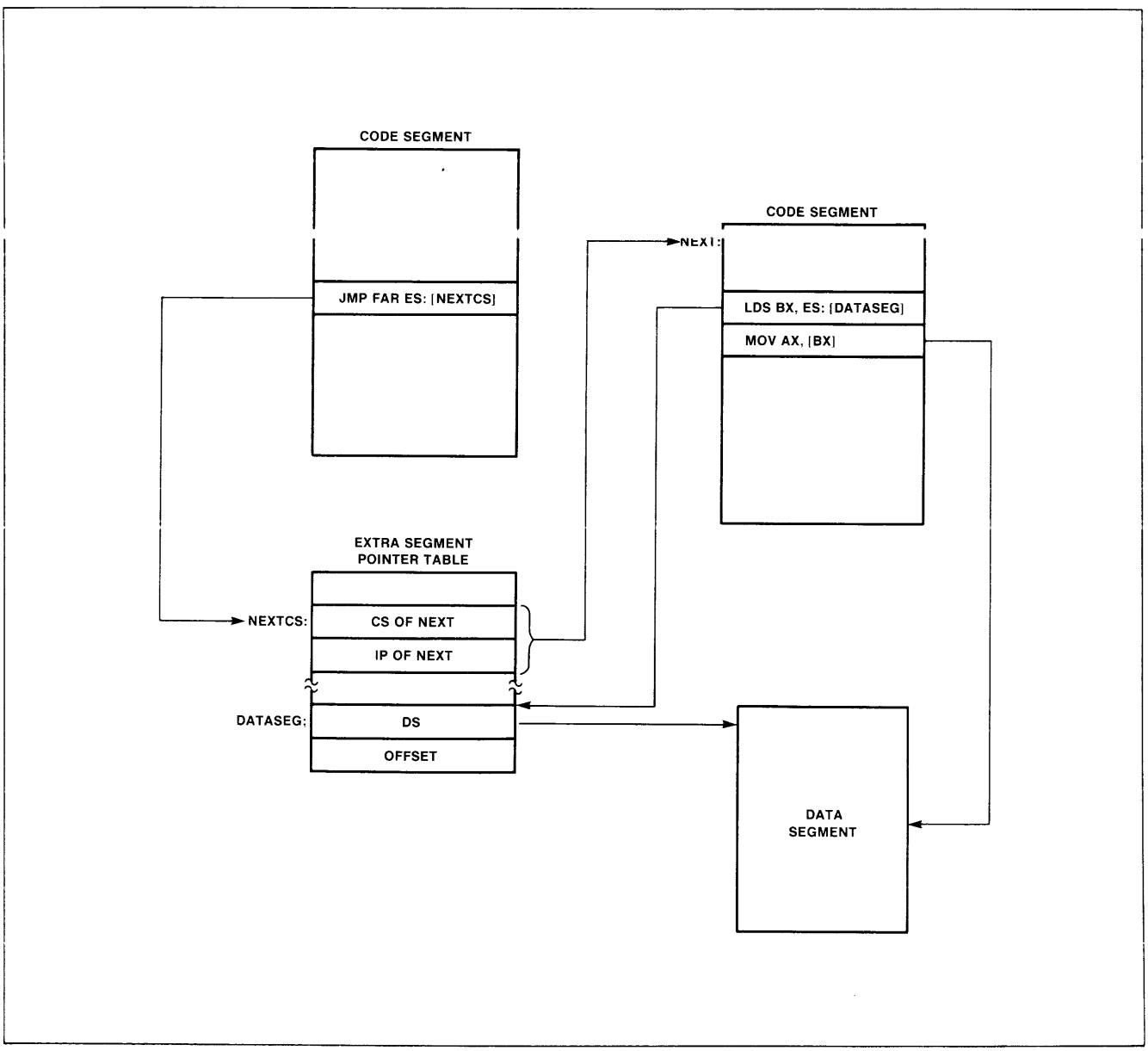

Figure A2. Common Table of Pointers

APPENDIX B

MONITOR LISTINGS

```
      MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1371
----                      1372     CODE SEGMENT WORD PUBLIC 'CODE'
                          1373
                          1374
                          1375     ;       EQUATES FOR MULTIUSER OPERATING SYSTEM
                          1376
                          1377
  0002                    1378     INPORT EQU 0002H        ;8155 PORT B ADDRESS (INPUT).
  0000                    1379     PORTCNT EQU 00000H      ;8155 PORT CONTROL REGISTER ADDRESS.
  0001                    1380     INOUT EQU 001H          ;CONFIGURATION DATA FOR 8155 PORTS.
  0007                    1381     BITS EQU 7
  F000                    1382     OUTPT2 EQU 0F000H       ;8755A PORT A ADDRESS (OUTPUT).
  F001                    1383     INPRT2 EQU 0F001H       ;8755A PORT B ADDRESS (INPUT).
  F002                    1384     OUTDR  EQU 0F002H       ;8755A PORT A DIRECTION CONTROL REG ADDRESS.
  F003                    1385     INDR   EQU 0F003H       ;8755A PORT B DIRECTION CONTROL REG ADDRESS.
  0001                    1386     OUTPORT EQU 0001H       ;8155 PORT A ADDRESS (OUTPUT).
  0005                    1387     TIMHI  EQU 5            ;ADDRESS OF 8155 TIMER.
                          1388
                          1389 +1  $EJ
```

```
      MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1390
                          1391     ;        THIS ROUTINE INITIALIZES VARIABLES FOR EACH USER
                          1392     ;                INCLUDING STACK AREAS AND STATE CONDITIONS.
                          1393     ;
                          1394     ;
FB3F B8----         R     1395     INIT:    MOV AX, DGROUP   ;INITIALIZE AX TO 0.
FB42 8ED8                 1396              MOV DS, AX       ;INITIALIZE DATA, STACK AND EXTRA SEGMENTS
FB44 8EC0                 1397              MOV     ES,AX    ;TO MEMORY LOCATION 0.
FB46 8ED0                 1398              MOV     SS,AX
FB48 BCFF1390       R     1399              MOV     SP,OFFSET(STK)
FB4C BF0010               1400              MOV     DI,1000H                ;SET DI TO USER1 VARIABLE AREA.
FB4F B91F00               1401              MOV     CX,1FH
FB52 F3                   1402              REP     STOSB                   ;CLEAR CONTROL VARIABLES.
FB53 AA
FB54 8926A710       R     1403              MOV     STACKP,SP               ;INITIALIZE COPY OF STACK POINTER.
FB58 FF0E0610       R     1404              DEC     M1
FB5C BE0010               1405              MOV     SI,1000H
FB5F BF0014               1406              MOV     DI,1400H                ;AX = OFFSET FOR USER 2.
FB62 B91F00               1407              MOV     CX,1FH
FB65 F3                   1408              REP     MOVSB                   ;COPY USER1 VARIABLES INTO USER2
FB66 A4
                          1409                                              ;CONTROL VARIABLE STORAGE SPACE.
                          1410
                          1411
                          1412     ;        INITIALIZE INTERRUPT VECTOR TABLE
                          1413
FB67 2EC706000088F5 R     1414              MOV CGROUP:DZO, OFFSET QHOW     ;DIVIDE BY ZERO INTERRUPT.
FB6E 2EC7060200---- R     1415              MOV CGROUP:DZS,CODE
FB75 2EC70608000AFC R     1416              MOV CGROUP:TOO, OFFSET TIMOUT   ;TIMER INTERRUPT.
FB7C 2EC7060A00---- R     1417              MOV CGROUP:TOS,CODE
                          1418
                          1419 +1  $EJ
```

```
       MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1420     ;       INITIALIZE THE I/O PORTS FOR TERMINAL I/O
                          1421
FB83 BA0000               1422             MOV DX,PORTCNT          ;LOAD ADDR OF 8155 COMMAND REG.
FB86 B001                 1423             MOV AL,INOUT            ;CONFIGURE PORTS:  PA=OUTPUT.
                          1424             ;                                  PB=INPUT.
FB88 EE                   1425             OUT DX,AL
FB89 BA0100               1426             MOV DX,OUTPORT          ;LOAD ADDR OF 8155 PORT A.
FB8C B0FF                 1427             MOV AL,0FFH             ;LOAD PORT WITH FFH TO BEGIN
                          1428             ;TRANSMITTING STOP BITS TO THIS
                          1429             ;TERMINAL.
FB8E EE                   1430             OUT DX,AL
FB8F BA03F0               1431             MOV DX,INDR             ;SET UP USER2'S I/O PORTS ON THE 8755A.
FB92 33C0                 1432             XOR AX,AX               ;ZERO OUT AX TO ESTABLISH PORT B
                          1433             ;AS AN INPUT PORT.
FB94 EE                   1434             OUT DX,AL               ;OUTPUT TO PORT B CONTROL REGISTER.
FB95 4A                   1435             DEC DX                  ;ADJUST DX TO INITIALIZE THE PORT A
                          1436             ;CONTROL REGISTER.
FB96 B0FF                 1437             MOV AL,0FFH             ;MAKE AL ALL ONES TO ESTABLISH PORT A
                          1438             ;AS AN OUTPUT PORT.
FB98 EE                   1439             OUT DX,AL               ;OUTPUT TO PORT A CONTROL REGISTER.
FB99 4A                   1440             DEC DX
FB9A 4A                   1441             DEC DX                  ;SET DX TO THE ADDRESS OF PORT A.
FB9B EE                   1442             OUT DX,AL               ;OUTPUT STOP BITS TO THE TERMINAL.
                          1443
                          1444
                          1445     ;       INITIALIZE REGISTERS FOR USER1 AND THE STACK FOR USER2
                          1446
FB9C B84000               1447             MOV AX,040H             ;SET AX TO USER2 SEGMENT VALUE.
FB9F 8ED0                 1448             MOV SS,AX               ;LOAD SEGMENT REGISTERS FOR USER2
FBA1 BCFF1390        R    1449             MOV SP,OFFSET(STK)
FBA5 8ED8                 1450             MOV DS,AX
FBA7 8EC0                 1451             MOV ES,AX
FBA9 33C0                 1452             XOR AX,AX               ;CLEAR AX AND INITIALIZE REGISTERS
FBAB 8BD8                 1453             MOV BX,AX               ;FOR USER2.
FBAD 8BC8                 1454             MOV CX,AX
FBAF 8BD0                 1455             MOV DX,AX
FBB1 8BE8                 1456             MOV BP,AX
FBB3 8BF0                 1457             MOV SI,AX
FBB5 8BF8                 1458             MOV DI,AX
FBB7 9C                   1459             PUSHF                   ;INITALIZE FLAG IMAGE FOR USER2
FBB8 0E                   1460             PUSH CS                 ;INITIALIZE CODE AND IP VALUES
FBB9 B890F5          R    1461             MOV AX,OFFSET(CGROUP:START)
FBBC 50                   1462             PUSH AX                 ;TO POINT TO THE START OF TINY BASIC
FBBD E8BB01               1463             CALL SVREG              ;GO PUSH ALL REGISTERS ONTO USER2'S
                          1464             ;STACK AND SWITCH TO USER1'S STACK.
                          1465             ;THIS MECHANISM INITIALIZES THE REGISTERS
                          1466             ;FOR USER1 AND INITIALIZES THE STACK FOR
                          1467             ;USER2.
                          1468 +1  $EJ
```

```
       MACRO ASSEMBLER    MTBASC


LOC  OBJ                  LINE     SOURCE

                          1469
                          1470
                          1471     ;        INITIALIZATION OF THE 8155 TIMER
                          1472
                          1473
                          1474
FBC0 BA0500               1475             MOV DX,TIMHI            ;SET TIMER COUNT AND MODE IN THE 8155 TIMER.
FBC3 B0CA                 1476             MOV AL,0CAH             ;MODE=4 (11, AUTO RELOAD AND SINGLE PULSE
                          1477                                     ;OUTPUT ON TERMINAL COUNT) AND COUNT=0ADAH.
FBC5 EE                   1478             OUT DX,AL               ;SET HIGH BYTE.
FBC6 4A                   1479             DEC DX
FBC7 B0DA                 1480             MOV AL,0DAH             ;SET LOW BYTE.  TIME DELAY = 1.111MS.
FBC9 EE                   1481             OUT DX,AL
FBCA 33D2                 1482             XOR DX,DX               ;SET PORT CONTROL TO START COUNTING.
FBCC B0C1                 1483             MOV AL,0C1H
FBCE EE                   1484             OUT DX,AL
FBCF E9BEF9               1485             JMP START               ;JUMP TO TINY BASIC TO START EXECUTION FOR
                          1486                                     ;USER1.
                          1487 +1  $EJ
```

```
      MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1488
                          1489     ;        CHARACTER INPUT AND CHARACTER OUTPUT ROUTINES
                          1490     ;
                          1491     ;        THESE ROUTINES TRANSFER THE USER INTO INPUT OR OUTPUT MODE
                          1492     ;        AND SUSPENDS TINY BASIC EXECUTION FOR THIS USER
                          1493     ;        UNTIL THE I/O IS COMPLETE.
                          1494     ;
                          1495     ;
FBD2 B002                 1496     CI:      MOV AL,2                ;SET INPUT MODE STATUS.
FBD4 C606A9100090      R  1497              MOV INCYCL,0            ;RESET INPUT CYCLES.
FBDA A2AE10            R  1498     COMP:    MOV STATUS,AL           ;SAVE STATUS.
FBDD 9C                   1499              PUSHF                   ;SET UP STACK FOR IRET.
FBDE 0E                   1500              PUSH CS
FBDF B8E9FB            R  1501              MOV AX,OFFSET(CGROUP:IORTI)     ;FORCE USER TO RETURN TO IORTI
                          1502                                              ;UNTIL INPUT IS COMPLETE.
FBE2 50                   1503              PUSH AX
FBE3 E89501               1504              CALL SVREG              ;SAVE REGISTERS FOR NORMAL RETURN.
FBE6 EB5E90               1505              JMP USER?               ;GO SWITCH TO OTHER USER.
FBE9 EBFE                 1506     IORTI:   JMP IORTI               ;LOOPS TO ITSELF UNTIL TIMEROUT.
FBEB A0AD10            R  1507     CIRT:    MOV AL,BYTEIN           ;RETURNS HERE WHEN CI HAS 7 BITS, LEAVE
                          1508                                      ;CHARACTER IN AL.
FBEE C3                   1509              RET                     ;RETURN TO TINY BASIC FOR THIS USER WHEN
                          1510                                      ;THE CHARACTER IS RECEIVED.
                          1511
                          1512
                          1513
FBEF 50                   1514     CO:      PUSH AX                 ;SAVE REGISTERS
FBF0 0D000F               1515              OR AX,0F00H             ;PUT STOP BITS IN WITH THE CHARACTER TO
                          1516                                      ;TO BE OUTPUT.
FBF3 D1E0                 1517              SAL AX,1                ;SHIFT LEFT TO SET UP START BIT (0) IN
                          1518                                      ;THE LOW ORDER BIT POSITION.
FBF5 A3AB10            R  1519              MOV WORDOT,AX           ;TRANSFER CHARACTER TO TEMPORARY STORAGE
                          1520                                      ;FOR OUTPUT TO THE TERMINAL.
FBF8 C606AA100090      R  1521              MOV OUTCYC,0            ;RESET OUTCYCLES AND BIT COUNT.
FBFE C606AF100090      R  1522              MOV CONNT,0
FC04 B001                 1523              MOV AL,1                ;SET STATUS TO OUTPUT MODE.
FC06 EBD2                 1524              JMP COMP                ;GO SAVE STATUS, SETUP STACK FOR IRET
                          1525                                      ;TO IORTI AND SWITCH TO THE OTHER USER.
FC08 58                   1526     CORT:    POP AX                  ;STORE THE AX REG.
FC09 C3                   1527              RET                     ;RETURN TO TINY BASIC FOR THIS USER WHEN
                          1528                                      ;THE CHARACTER HAS BEEN TRANSMITTED.
                          1529
                          1530
                          1531 +1  $EJ
```

```
        MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1532
                          1533
                          1534     ;                THIS IS THE TIMER INTERRUPT SERVICE ROUTINE WHICH
                          1535     ;                PERFORMS TERMINAL INPUT AND OUTPUT AND MULTIPLEXES
                          1536     ;                THE CPU TIME BETWEEN THE TWO USERS.
                          1537
                          1538
                          1539
                          1540     TIMOUT:  CALL SVREG              ;SAVE REGISTERS OF CURRENT USER.
                          1541              MOV DX,INPORT
                          1542              IN AL,DX                ;GET INPUT FROM USER1.
                          1543              MOV AH,AL               ;SAVE IN AH.
                          1544              MOV DX,INPRT2
                          1545              IN AL,DX                ;GET INPUT FROM USER2.
                          1546              PUSH AX                 ;SAVE SAMPLES FROM BOTH USERS FOR SECOND
                          1547                                      ;SAMPLE TIME.
                          1548              MOV CX,AX               ;INPUT DATA, SAVE IN CX
                          1549              MOV DX,OUTPORT
                          1550              CALL OUTWORD            ;OUTPUT BIT FOR USER 1
                    R     1551              MOV STACKP,SP           ;SAVE USER1 STACK POINTER.
                          1552              MOV DX,OUTPT2           ;LOAD I/O ADDRESS FOR USER2 OUTPUT.
                          1553              MOV AX,0040H            ;SWITCH TO USER2.
                          1554              MOV SS,AX               ;SET STACK, DATA AND EXTRA SEGMENTS TO
                          1555                                      ;USER2.
                          1556              MOV ES,AX
                          1557              MOV DS,AX
                    R     1558              MOV SP,STACKP           ;LOAD USER2 STACK POINTER.
                          1559              CALL OUTWORD            ;OUTPUT BIT FOR USER 2
                          1560              MOV DX,INPRT2           ;TAKE SECOND INPUT DATA SAMPLE FOR
                          1561                                      ;USER2.  CX HAS FIRST SAMPLE.
                          1562              CALL INBYTE             ;ASSEMBLE INPUT.
                          1563              POP CX                  ;RESTORE FIRST SAMPLE TAKEN FOR USER1.
                          1564              MOV CL,CH               ;RESTORE PORT 1'S BYTE.
                          1565              MOV DX,INPORT           ;TAKE SECOND SAMPLE FOR USER1.
                          1566              CALL INBYTE             ;ASSEMBLE INPUT.
                          1567
                          1568
                          1569
                    R     1570     USER?:   MOV AL,STATUS           ;CHECK USER 1 STATUS.
                          1571              AND AL,03H
                          1572              JZ CKU2                 ;JUMP IF USER1 NOT IN CO OR CI,
                          1573                                      ;CHECK USER 2.
                          1574              MOV AX,0040H            ;USER 1 IN CO OR CI, SET SEGMENTS
                          1575              JMP PRET1               ;FOR USER2 SINCE USER1 IS IN I/O.
                    R     1576     CKU2:    MOV AL,STATS2           ;CHECK USER 2 STATUS.
                          1577              AND AL,03H
                          1578              JZ  SWUS                ;JUMP IF USER2 ALSO IS NOT IN CO OR CI.
                          1579              XOR AX,AX               ;BOTH USERS IN I/O, DEFAULT TO USER1.
                          1580              JMP PRET1
                          1581     SWUS:    MOV AX,CS:STACKS        ;SWITCH USERS:  GET CURRENT STACK SEGMENT.
                          1582              XOR AX,0040H            ;SWITCH STACKS.
                          1583     PRET1:   MOV SS,AX               ;LOAD SEGMENT REGISTERS.
                          1584              MOV DS,AX
                    R     1585              MOV SP,STACKP
```

```
       MACRO ASSEMBLER    MTBASC


LOC  OBJ                  LINE      SOURCE

FC6E 1F                   1586              POP DS                  ;RESTORE USERS MACHINE STATE FOR TINY
FC6F 07                   1587              POP ES                  ;BASIC EXECUTION.
FC70 5D                   1588              POP BP
FC71 5F                   1589              POP DI
FC72 5E                   1590              POP SI
FC73 5A                   1591              POP DX
FC74 59                   1592              POP CX
FC75 5B                   1593              POP BX
FC76 58                   1594              POP AX
FC77 CF                   1595              IRET                    ;RETURN TO TINY BASIC FOR ACTIVE USER
                          1596                                      ;OR IORTI IF BOTH USERS ARE IDLE (IN I/O).
                          1597
                          1598 +1  $EJ
```

```
       MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1599     ;                THIS ROUTINE OUTPUTS THE CHARACTERS TO THE
                          1600     ;                TERMINAL A BIT AT A TIME.  IF NO CHARACTERS ARE BEING
                          1601     ;                TRANSMITTED, A STOP BIT IS SENT.
                          1602
FC78 A0AF10          R    1603     OUTWORD: MOV AL,CONNT            ;ONLY OUTPUT EVERY 3RD CYCLE.
FC7B 2403                 1604              AND     AL,03H
FC7D 7514                 1605              JNZ     OUT1            ;NOT ON THIS CYCLE.
FC7F A1AB10          R    1606              MOV AX,WORDOT           ;LOAD WORD OUT.
FC82 8A1EAE10        R    1607              MOV BL,STATUS           ;LOAD STATUS BYTE.
FC86 80CBFE               1608              OR BL,0FEH
FC89 F6D3                 1609              NOT BL                  ;FORCE THE
FC8B 0AC3                 1610              OR AL,BL                ;LOW ORDER BIT TO BE A STOP BIT
FC8D EE                   1611              OUT DX,AL               ;IF NOT IN CO ROUTINE.
FC8E D1F8                 1612              SAR AX,1                ;SHIFT FOR NEXT BIT TO OUTPUT.
FC90 A3AB10          R    1613              MOV WORDOT,AX           ;SAVE FOR NEXT BIT TIME.
FC93 FE06AF10        R    1614     OUT1:    INC     CONNT           ;INCREMENT COUNT.
FC97 803EAF1003      R    1615              CMP     CONNT,3
FC9C 7506                 1616              JNE     OTRT            ;JUMP IF STILL TRANSMITTING THIS BIT.
FC9E C606AF100090    R    1617              MOV     CONNT,0         ;RESET COUNT IF EQUAL TO 3.  CONNT=0
                          1618                                      ;INDICATES TIME TO BEGIN TRANSMITTING
FCA4 C3                   1619     OTRT:    RET                     ;THE NEXT BIT.
                          1620
                          1621     ;                THIS ROUTINE CONSTRUCTS THE BIT RECEIVED AND
                          1622     ;                ASSEMBLES THE BITS INTO CHARACTERS.
                          1623
FCA5 8A1EAE10        R    1624     INBYTE:  MOV BL,STATUS           ;LOAD STATUS.
FCA9 8AFB                 1625              MOV BH,BL               ;AND SAVE FOR LATER.
FCAB 80E301               1626              AND BL,01H              ;TEST FOR OUTPUT STATUS.
FCAE 743A                 1627              JZ CKIN
FCB0 803EAF1000      R    1628              CMP CONNT,0             ;USER IN OUTPUT MODE.
FCB5 7520                 1629              JNZ BRET                ;RETURN IF STILL TRANSMITTING THIS BIT.
FCB7 FE06AA10        R    1630              INC OUTCYC              ;INCREMENT NUMBER OF BITS TRANSMITTED.
FCBB 803EAA100A      R    1631              CMP OUTCYC,10           ;TEST IF ALL HAVE BEEN SENT INCLUDING
                          1632                                      ;START AND STOP BITS.
FCC0 7515                 1633              JNE BRET                ;RETURN IF STILL TRANSMITTING THIS
                          1634                                      ;CHARACTER.
FCC2 C606AE100090    R    1635              MOV STATUS,00H          ;IF ALL BITS HAVE BEEN TRANSMITTED,
FCC8 BB08FC          R    1636              MOV BX,OFFSET(CGROUP:CORT)      ;RESET STATUS AND MODIFY THE
FCCB 8926A710        R    1637     RSST:    MOV STACKP,SP           ;USERS STACK TO RETURN TO CORT AND TINY
                          1638                                      ;BASIC RATHER THAN THE WAIT LOOP AT IORTI.
FCCF 83C416               1639              ADD SP,22
FCD2 53                   1640              PUSH BX                 ;MOFIFY IP ON THE STACK TO RETURN TO
                          1641                                      ;PROPER ROUTINE.
FCD3 8B26A710        R    1642              MOV SP,STACKP           ;RESTORE STACK POINTER.
FCD7 59                   1643     BRET:    POP CX                  ;SAVE IP SO IT CAN BE RESTORED AFTER
                          1644                                      ;STACK SWITCH.
FCD8 8926A710        R    1645              MOV STACKP,SP
FCDC 33C0                 1646              XOR AX,AX               ;SWITCH TO USER1 IN CASE THIS WAS USER2.
FCDE 8ED0                 1647              MOV SS,AX
FCE0 8ED8                 1648              MOV DS,AX
FCE2 8EC0                 1649              MOV ES,AX
FCE4 8B26A710        R    1650              MOV SP,STACKP
FCE8 51                   1651              PUSH CX                 ;RESTORE RETURN ADDRESS
FCE9 C3                   1652              RET                     ;AND RETURN.
```

```
      MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1653
                          1654     ;                THESE ROUTINES ARE USED BY INBYTE TO CONTRUCT
                          1655     ;                THE INPUT CHARACTERS.
                          1656
FCEA 8ADF                 1657     CKIN:   MOV      BL,BH            ;SEE IF IN INPUT MODE.
FCEC 80E302               1658             AND      BL,02H
FCEF 74E6                 1659             JZ       BRET             ;RETURN IF NOT IN INPUT MODE.
FCF1 80E704               1660             AND      BH,04H           ;SEE IF STILL WAITING FOR START BIT.
FCF4 7468                 1661             JZ       WAITST           ;JMP IF STILL WAITING.
FCF6 80E180               1662             AND      CL,80H           ;TEST THE BIT SAMPLED THE FIRST TIME.
FCF9 7407                 1663             JZ       CK1              ;IF BIT-IN IS A 0, SHIFT LEFT
FCFB D10EA510        R    1664             ROR      BITSS,1          ;ELSE SHIFT BYTE RIGHT.
FCFF EB0590               1665             JMP      CK2
FD02 D106A510        R    1666     CK1:    ROL      BITSS,1
FD06 EC                   1667     CK2:    IN       AL,DX            ;TAKE A SECOND SAMPLE.
FD07 2480                 1668             AND      AL,80H           ;TEST IT AND SHIFT BITSS RIGHT OR
FD09 7407                 1669             JZ       CK3              ;LEFT ACCORDINGLY.
FD0B D10EA510        R    1670             ROR      BITSS,1
FD0F EB0590               1671             JMP      CK4
FD12 D106A510        R    1672     CK3:    ROL      BITSS,1
FD16 FE06AF10        R    1673     CK4:    INC      CONNT            ;UP COUNT BY 1.
FD1A 803EAF1003      R    1674             CMP      CONNT,3          ;TEST IF DONE SAMPLING THIS BIT WINDOW.
FD1F 75B6                 1675             JNZ      BRET             ;IF NOT THIRD COUNT, WAIT FOR MORE
FD21 C606AF100090    R    1676             MOV      CONNT,0          ;ELSE RESET CONNT.
FD27 A1A510          R    1677             MOV      AX,BITSS         ;BRING IN VOTE.
FD2A C706A510FF00    R    1678             MOV      BITSS,00FFH      ;RESET VOTE COUNTER.
FD30 80E480               1679             AND      AH,80H           ;SAVE THE RESULTING BIT.
FD33 D02EAD10        R    1680             SHR      BYTEIN,1         ;MAKE ROOM FOR NEXT BIT.
FD37 0826AD10        R    1681             OR       BYTEIN,AH        ;OR ON NEXT BIT (MAJORITY RULE).
FD3B FE06A910        R    1682             INC      INCYCL           ;ALL BITS IN?
FD3F 803EA91009      R    1683             CMP      INCYCL,9
FD44 7591                 1684             JNE      BRET             ;JUMP IF NOT AND WAIT FOR MORE.
FD46 C606A9100090    R    1685             MOV      INCYCL,0         ;ALL RECEIVED, RESET INCYCLE.
FD4C 8026AD107F90    R    1686             AND      BYTEIN,7FH       ;PREPARE BYTE IN FOR RETURN TO PROGRAM
FD52 C606AE100090    R    1687             MOV      STATUS,0         ;RESET STATUS
FD58 BBEBFB          R    1688             MOV      BX,OFFSET(CGROUP:CIRT)   ;SET BX TO RETURN LOCATION CIRT.
FD5B E96DFF               1689             JMP      RSST             ;GO MODIFY STACK FOR PROPER RETURN.
FD5E 80E180               1690     WAITST: AND      CL,80H           ;SEE IF THIS IS A START BIT
FD61 7403                 1691             JZ       SETST
FD63 E971FF               1692             JMP      BRET             ;NO START BIT.
FD66 C606AE100690    R    1693     SETST:  MOV      STATUS,6         ;START BIT FOUND, SET STATUS TO INPUT MODE.
FD6C C606AF100190    R    1694             MOV      CONNT,1          ;SET COUNT AS IF 0 HAS BEEN ENTERED (IT HAS)
FD72 C706A510FE01    R    1695             MOV      BITSS,01FEH      ;INITIALIZE BIT SAMPLE WORD.
FD78 E95CFF               1696             JMP      BRET
                          1697 +1  $EJ
```

```
       MACRO ASSEMBLER    MTBASC

LOC  OBJ                  LINE     SOURCE

                          1698
                          1699     ;               THIS ROUTINE SAVES THE STATE OF THE MACHINE FOR
                          1700     ;               THE CURRENT USER AND RETURNS TO USER1.
                          1701
FD7B 891E0010       R     1702     SVREG:  MOV BL1,BX              ;SAVE THE RETUNE ADDRESS ON THE STACK
FD7F 5B                   1703             POP BX                 ;IN THE BX REGISTER.
FD80 50                   1704             PUSH AX                ;SAVE THE CURRENT MACHINE STATE.
FD81 FF360010       R     1705             PUSH BL1
FD85 51                   1706             PUSH CX
FD86 52                   1707             PUSH DX
FD87 56                   1708             PUSH SI
FD88 57                   1709             PUSH DI
FD89 55                   1710             PUSH BP
FD8A 06                   1711             PUSH ES
FD8B 1E                   1712             PUSH DS
FD8C 8926A710       R     1713             MOV STACKP,SP          ;SAVE THE STACK POINTER.
FD90 8CD1                 1714             MOV CX,SS
FD92 33C0                 1715             XOR AX,AX              ;SWITCH TO USER1.
FD94 8ED0                 1716             MOV SS,AX              ;LOAD SEGMENT REGISTERS.
FD96 8EC0                 1717             MOV ES,AX
FD98 8ED8                 1718             MOV DS,AX
FD9A 8B26A710       R     1719             MOV SP,STACKP          ;LOAD THIS USERS STACK POINTER.
FD9E 2E890E0C00           1720             MOV CS:STACKS,CX       ;SAVE PREVIOUS USERS STACK SEGMENT.
FDA3 53                   1721             PUSH BX                ;RESTORE THE RETURN ADDRESS.
FDA4 C3                   1722             RET                    ;AND RETURN.
FDA5                      1723     LSTROM  LABEL BYTE
----                      1724     CODE ENDS
                          1725
                          1726
                          1727 +1  $EJ
```

```
       MACRO ASSEMBLER    MTBASC


LOC  OBJ                  LINE     SOURCE

                          1728
                          1729     ;               COME HERE AFTER RESET AND JUMP TO INITIALIZATION.
                          1730
                          1731
----                      1732     CODE1 SEGMENT WORD PUBLIC 'CODE'
FFF0                      1733     ORG OFFF0H
FFF0 EA3FFB----      R    1734             JMP FAR PTR INIT        ;JUMP TO INIT
----                      1735     CODE1 ENDS
                          1736
                          1737
                          1738     ;               DATA STRUCTURE DEFINITION
                          1739     ;
                          1740     ;       THIS SECTION DEFINES THE DATA STUCTURE WHICH CONTAINS THE
                          1741     ;       VARIABLES AND CONTROL INFORMATION FOR EACH USER IN THE SYSTEM.
                          1742     ;       NOTE THE STRUCTURE IS DEFINED ONLY ONCE SINCE ALL OFFSETS ARE
                          1743     ;       FORMED RELATIVE TO THE SEGMENT REGISTERS WHICH ARE TRANSPARENT
                          1744     ;       TO THE USERS.  THIS DEFINITION SERVES AS A TEMPLATE FOR CONTRUCTING
                          1745     ;       THE OFFSETS USED, AND IS REPLICATED FOR EACH USER.
                          1746
                          1747
                          1748
----                      1749     DATA    SEGMENT WORD PUBLIC 'DATA'
1000                      1750     ORG 1000H               ;TINY BASIC VARIABLES.
1000 ????                 1751     BL1     DW      ?
1002 ????                 1752     BL2     DW      ?
1004 0000                 1753     ZERO    DW      0
1006 FFFF                 1754     M1      DW      -1
1008 00                   1755     POLO    DB      0
1009 00                   1756     RICI    DB      0
100A 0000                 1757     CURRNT  DW      0
100C 00                   1758     Z       DB      0
100D 0000                 1759     STKGOS  DW      0
100F                      1760     VARNXT  LABEL   WORD
100F 0000                 1761     STKINP  DW      0
1011 0000                 1762     LOPVAR  DW      0
1013 0000                 1763     LOPINC  DW      0
1015 0000                 1764     LOPLMT  DW      0
1017 0000                 1765     LOPLN   DW      0
1019 0000                 1766     LOPPT   DW      0
101B 90F5            R    1767     RANPNT  DW      CGROUP:START
101D B010            R    1768     TXTUNF  DW      DGROUP:TXTBGN
                          1769
----                      1770     DATA    ENDS
                          1771
----                      1772     DATA2   SEGMENT WORD 'DATA'
1020                      1773             ORG 01020H              ;O.S.  VARIABLES FOR EACH USERS I/O.
                          1774
1020 (52                  1775     VARBGN  DB      2*26 DUP (?)
     ??
     )
1054 (1                   1776             DB      1 DUP (?)
     ??
     )

                          1777 +1  $EJ
```

```
      MACRO ASSEMBLER    MTBASC


LOC  OBJ                  LINE     SOURCE

1055 (80                  1778     BUFFER  DB      80 DUP (?)      ;LINE BUFFER FOR CRT I/O.
     ??
     )
10A5                      1779     BUFEND  LABEL   BYTE
10A5 ????                 1780     BITSS   DW      ?       ;SAMPLE STORAGE FOR INCOMING BIT RECOGNITION.
10A7 ????                 1781     STACKP  DW      ?       ;USER STACK POINTER WHEN IDLE.
10A9 ??                   1782     INCYCL  DB      ?       ;INPUT BIT COUNT.
10AA ??                   1783     OUTCYC  DB      ?       ;SERIAL OUTPUT BIT COUNT.
10AB ????                 1784     WORDOT  DW      ?       ;STORAGE FOR CHARACTERS BEING DISASSEMBLED.
10AD ??                   1785     BYTEIN  DB      ?       ;STORAGE FOR CHARACTERS BEING ASSEMBLED.
10AE ??                   1786     STATUS  DB      ?       ;I/O MODE STATUS.
10AF ??                   1787     CONNT   DB      ?       ;BIT INPUT SAMPLE OR OUTPUT TIMING COUNT.
14AE                      1788             ORG 14AEH
14AE ??                   1789     STATS2  DB      ?       ;I/O MODE STATUS FOR USER 2.
----                      1790     DATA2   ENDS
                          1791
----                      1792     STACK   SEGMENT WORD STACK 'STACK'
1300                      1793             ORG 01300H
1300                      1794     STKLMT  LABEL   BYTE
1300 (255                 1795             DB      OFFH DUP (?)
     ??
     )
13FF                      1796     STK     LABEL   WORD
                          1797
----                      1798     STACK   ENDS
  FB3F                    1799     END     INIT

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

intel

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

U.S. AND CANADIAN SALES OFFICES

September 1980

ALABAMA

Intel Corp.
303 Williams Avenue, S.W.
Suite 1422
Huntsville 35801
Tel: (205) 533-9353

ARIZONA

Intel Corp.
10210 N. 25th Avenue, Suite 11
Phoenix 85021
Tel: (602) 997-9695

BFA
4426 North Saddle Bag Trail
Scottsdale 85251
Tel: (602) 994-5400

CALIFORNIA

Intel Corp.
7670 Opportunity Rd.
Suite 135
San Diego 92111
Tel: (714) 268-3563

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
5530 Corbin Avenue
Suite 120
Tarzana 91356
Tel. (213) 986-9510
TWX: 910-495-2045

Intel Corp.*
3375 Scott Blvd.
Santa Clara 95051
Tel: (408) 987-8086
TWX: 910-339-9279
910-338-0255

Earle Associates, Inc.
4617 Ruffner Street
Suite 202
San Diego 92111
Tel: (714) 278-5441

Mac-I
P.O. Box 1420
Cupertino 95014
Tel: (408) 257-9880

Mac-I
558 Valley Way
Calaveras Business Park
Milpitas 95035
Tel: (408) 946-8885

Mac-I
P.O. Box 8763
Fountain Valley 92708
Tel: (714) 839-3341

Mac-I
1321 Centinela Avenue
Suite 1
Santa Monica 90404
Tel: (213) 829-4797

Mac-I
20121 Ventura Blvd., Suite 240E
Woodland Hills 91364
Tel: (213) 347-5900

COLORADO

Intel Corp.*
650 S. Cherry Street
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
Peacock Alley
36 Padanaram Road
Danbury 06810
Tel: (203) 792-8366
TWX: 710-456-1199

FLORIDA

Intel Corp.
1001 N.W. 62nd Street, Suite 406
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

FLORIDA (cont.)

Intel Corp.
5151 Adanson Street, Suite 203
Orlando 32804
Tel: (305) 628-2393
TWX: 810-853-9219

GEORGIA

Intel Corp.
3300 Holcomb Bridge Rd.
Norcross 30092

ILLINOIS

Intel Corp.*
2550 Golf Road, Suite 815
Rolling Meadows 60008
Tel: (312) 981-7200
TWX: 910-651-5881

Technical Representatives
1502 North Linde Street
Bloomington 61701
Tel: (309) 829-8080

INDIANA

Intel Corp.
9101 Wesleyan Road
Suite 204
Indianapolis 46268
Tel: (317) 299-0623

IOWA

Technical Representatives, Inc.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52405
Tel: (319) 393-5510

KANSAS

Intel Corp.
9393 W. 110th St., Ste. 265
Overland Park 66210
Tel: (913) 642-8080

Technical Representatives, Inc.
8245 Nieman Road, Suite 100
Lenexa 66214
Tel: (913) 888-0212, 3, & 4
TWX: 910-749-6412

Technical Representatives, Inc.
360 N. Rock Road
Suite 4
Wichita 67206
Tel: (316) 681-0242

MARYLAND

Intel Corp.*
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Mesa Inc.
16021 Industrial Dr.
Gaithersburg 20760
Tel: (301) 948-4350

MASSACHUSETTS

Intel Corp.*
27 Industrial Ave.
Chelmsford 01824
Tel: (617) 667-8126
TWX: 710-343-6333

EMC Corp.
381 Elliot Street
Newton 02164
Tel: (617) 244-4740
TWX: 922531

MICHIGAN

Intel Corp.*
26500 Northwestern Hwy.
Suite 401
Southfield 48075
Tel: (313) 353-0920
TWX: 810-244-4915

MINNESOTA

Intel Corp.
7401 Metro Blvd.
Suite 355
Edina 55435
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
502 Earth City Plaza
Suite 121
Earth City 63045
Tel: (314) 291-1990

Technical Representatives, Inc.*
502 Earth City Plaza
Suite 201
Earth City 63045
Tel: (314) 291-0001

Technical Representatives, Inc.*
VSW Bldg. Suite 560
406 W. 31st Street
Kansas City 64111
Tel: (816) 756-3575
TWX: 910-771-0025

NEW JERSEY

Intel Corp.*
Raritan Plaza
2nd Floor
Raritan Center
Edison 08817
Tel: (201) 225-3000
TWX: 710-480-6238

NEW MEXICO

BFA Corporation
1704 Moon N.E., Suite 7
Las Cruces 87112
Tel: (505) 523-0601
TWX: 910-983-0543

BFA Corporation
3705 Westerfield, N.E.
Albuquerque 87111
Tel: (505) 292-1212
TWX: 910-989-1157

NEW YORK

Intel Corp.*
300 Motor Pkwy.
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington St.
Poughkeepsie 12601
Tel: (914) 473-2303
TWX: 510-248-0060

Intel Corp.*
2255 Lyell Avenue
Lower Floor East Suite
Rochester 14606
Tel: (716) 254-6120
TWX: 510-253-7391

Measurement Technology, Inc.
159 Northern Boulevard
Great Neck 11021
Tel: (516) 482-3500

T-Squared
4054 Newcourt Avenue
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
2 E. Main
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA

Intel Corp.
154 Huffman Mill Rd.
Burlington 27215
Tel: (919) 584-3631

OHIO

Intel Corp.*
6500 Poe Avenue
Dayton 45415
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg., No. 300
28001 Chagrin Blvd.
Cleveland 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 324
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.*
275 Commerce Dr.
200 Office Center
Suite 300
Fort Washington 19034
Tel: (215) 542-9444
TWX: 510-661-2077

Intel Corp.*
201 Penn Center Boulevard
Suite 301W
Pittsburgh 15235
Tel: (412) 823-4970

Q.E.D. Electronics
300 N. York Road
Hatboro 19040
Tel: (215) 674-9600

TEXAS

Intel Corp.*
2925 L.B.J. Freeway
Suite 175
Dallas 75234
Tel: (214) 241-9521
TWX: 910-860-5617

Intel Corp.*
6420 Richmond Ave.
Suite 280
Houston 77057
Tel: (713) 784-3400
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713) 988-9421

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

WASHINGTON

Intel Corp.
Suite 114, Bldg. 3
1603 116th Ave. N.E.
Bellevue 98005
Tel: (206) 453-8086
TWX: 910-443-3002

WISCONSIN

Intel Corp.
150 S. Sunnyslope Rd.
Brookfield 53005
Tel: (414) 784-9060

CANADA

Intel Semiconductor Corp.*
Suite 233, Bell Mews
39 Highway 7, Bells Corners
Ottawa, Ontario K2H 8R2
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor Corp.
50 Galaxy Blvd.
Unit 12
Rexdale, Ontario
M9W 4Y5
Tel: (416) 675-2105
TELEX: 06983574

Multilek, Inc.*
15 Grenfell Crescent
Ottawa, Ontario K2G 0G3
Tel: (613) 226-2365
TELEX: 053-4585

Multilek, Inc.
Toronto
Tel: (416) 245-4622

Multilek, Inc.
Montreal
Tel: (514) 481-1350

*Field Application Location

intel
3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

U.S. AND CANADIAN DISTRIBUTORS

September 1980

ALABAMA

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210

†Pioneer/Huntsville
1207 Putman Drive NW
Huntsville 35805
Tel: (205) 837-9033
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
2615 S. 21st Street
Phoenix 85034
Tel: (602) 275-7851

†Wyle Distribution Group
8155 N. 24th Avenue
Phoenix 85021
Tel: (602) 995-9185
TWX: 910-951-4282

CALIFORNIA

Arrow Electronics, Inc.
9511 Ridge Haven Court
San Diego 92123
Tel: (714) 565-4800

Arrow Electronics, Inc.
720 Palomar Avenue
Sunnyvale, California 94086
Tel: (408) 739-3011
TWX: 910-339-9371

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Dr.
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
8917 Complex Drive
San Diego 92123
Tel: (714) 571-7923
TWX: 910-335-1216

†Hamilton/Avnet Electronics
10912 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2193
TWX: 910-340-6364 or 7073

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4100
TWX: 910-595-2638

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-3826
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
9525 Chesapeake Dr.
San Diego 92123
Tel: (714) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95052
Tel: (408) 727-2500
TWX: 910-338-0451 or 0296

Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 641-1611

COLORADO

†Wyle Distribution Group
6777 E. 50th Avenue
Commerce City 80022
Tel: (303) 287-9611
TWX: 910-931-0510

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 534-1212
TWX: 910-931-0510

COLORADO (cont.)

Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-WYLE
TWX: 910-931-0510

CONNECTICUT

†Arrow Electronics
12 Beaumont Road
Wallingford 06512
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Harvey Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373
TWX: 710-393-6770

FLORIDA

†Arrow Electronics
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics
115 Palm Bay Road, NW
Suite 10, Bldg. 200
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6800 Northwest 20th Ave.
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-955-3097

Hamilton/Avnet Electronics
3197 Tech. Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

†Pioneer/Orlando
6220 S. Orange Blossom Trail
Suite 412
Orlando 32809
Tel: (305) 859-3600
TWX: 810-850-0177

GEORGIA

Arrow Electronics
2979 Pacific Drive
Norcross 30071
Tel: (404) 449-8252
TWX: 810-757-4213

†Hamilton/Avnet Electronics
6700 I-85 Access Road, No. 11
Suite 1E
Norcross 30071
Tel: (404) 448-0800

ILLINOIS

Arrow Electronics
492 Lunt Avenue
P.O. Box 94248
Schaumburg 60172
Tel: (312) 893-9420
TWX: 910-222-1807

†Hamilton/Avnet Electronics
3901 No. 25th Avenue
Schiller Park 60176
Tel: (312) 678-6310
TWX: 910-227-0060

Pioneer/Chicago
1551 Carmen Drive
Elk Grove 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333

Pioneer/Indiana
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivira Road
Overland Park 66215
Tel: (913) 888-8900

†Component Specialties, Inc.
8369 Nieman Road
Lenexa 66214
Tel: (913) 492-3555

MARYLAND

Arrow Electronics, Inc.
4801 Benson Avenue
Baltimore 21227
Tel: (301) 247-5200

†Hamilton/Avnet Electronics
7235 Standard Drive
Hanover 21076
Tel: (301) 796-5684
TWX: 710-862-1861

†Pioneer/Washington
9100 Gaither Road
Gaithersburg 20760
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel: (617) 273-7500
TWX: 710-393-0382

†Arrow Electronics
96D Commerce Way
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

Harvey/Boston
44 Hartwell Ave.
Lexington 02173
Tel: (617) 861-9200
TWX: 710-326-6617

MICHIGAN

†Arrow Electronics
3810 Varsity Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

MINNESOTA

†Arrow Electronics
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-756-2726

†Industrial Components
5229 Edina Industrial Blvd.
Minneapolis 55435
Tel: (612) 831-2666
TWX: 910-756-3153

†Hamilton/Avnet Electronics
7449 Cahill Road
Edina 55435
Tel: (612) 941-3801
TWX: 910-576-2720

MISSOURI

†Hamilton/Avnet Electronics
13743 Shoreline Ct.
Earth City, 63045
Tel: (314) 344-1200
TWX: 910-762-0606

NEW HAMPSHIRE

†Arrow Electronics
1 Perimeter Drive
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

NEW JERSEY

†Arrow Electronics
Pleasant Valley Avenue
Moorestown 08057
Tel: (215) 928-1800
TWX: 710-897-0829

†Arrow Electronics
285 Midland Avenue
Saddle Brook 07662
Tel: (201) 797-5800
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Ave.
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0100
TWX: 710-897-1405

†Harvey Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 227-1262
TWX: 710-734-4382

Hamilton/Avnet Electronics
10 Industrial Road
Fairfield 07006
Tel: (201) 575-3390
TWX: 710-734-4438

NEW MEXICO

†Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

†Hamilton/Avnet Electronics
2524 Baylor Drive, S.E.
Albuquerque 87119
Tel: (505) 765-1500

NEW YORK

†Arrow Electronics
3000 South Winton Road
Rochester 14623
Tel: (716) 275-0300
TWX: 510-253-4766

†Arrow Electronics
7705 Maltlage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics
20 Oser Avenue
Hauppauge 11787
Tel: (516) 231-1000
TWX: 510-227-6623

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
16 Corporate Circle
E. Syracuse 13057
Tel: (315) 437-2641

†Hamilton/Avnet Electronics
5 Hub Drive
Melville, Long Island 11746
Tel: (516) 454-6000
TWX: 510-252-0893

†Microcomputer System Technical Demonstrator Centers

intel
3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

U.S. AND CANADIAN DISTRIBUTORS

September 1980

NEW YORK (cont.)

Harvey Electronics
P.O. Box 1208
Binghampton 13902
Tel: (607) 748-8211
TWX: 510-252-0893

†Harvey Electronics
60 Crossways Park West
Woodbury 11797
Tel: (516) 921-8700
TWX: 510-221-2184

Harvey/Rochester
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

Arrow Electronics
938 Burke Street
Winston-Salem 27102
Tel: (919) 725-8711
TWX: 510-922-4765

Pioneer/Carolina
106 Industrial Ave.
Greensboro 27406
Tel: (919) 273-4441
TWX: 510-925-1114

†Hamilton/Avnet Electronics
2803 Industrial Drive
Raleigh 27609
Tel: (919) 829-8030

OHIO

Arrow Electronics
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

Arrow Electronics
6238 Cochran Rd.
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

Arrow Electronics
10 Knollcrest Dr.
Cincinnati 45237
Tel: (513) 761-5432
TWX: 810-461-2670

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 910-340-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer/Dayton
1900 Troy Street
Dayton 45404
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer/Cleveland
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2210

OKLAHOMA

†Components Specialties, Inc.
7920 E. 40th Street
Tulsa 74145
Tel: (918) 664-2820
TWX: 910-845-2215

OREGON

†Almac/Stroum Electronics
8022 S.W. Nimbus, Bldg. 7
Beaverton 97005
Tel: (503) 641-9070

†Hamilton/Avnet Electronics
6024 SW Jean Rd.
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848

PENNSYLVANIA

†Arrow Electronics
4297 Greensburg Pike
Suite 3114
Pittsburgh 15221
Tel: (412) 351-4000

Pioneer/Pittsburgh
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

Pioneer/Delaware Valley
261 Gibraltar Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

Arrow Electronics
13715 Gamma Road
Dallas 75234
Tel: (214) 386-7500
TWX: 910-861-5495

Arrow Electronics, Inc.
10700 Corporate Drive, Suite 100
Stafford 77477
Tel: (713) 491-4100

Component Specialties Inc.
8222 Jamestown Drive
Suite 115
Austin 78758
Tel: (512) 837-8922
TWX: 910-874-1320

†Component Specialties, Inc.
10807 Shady Trail, Suite 101
Dallas 75220
Tel: (214) 357-6511
TWX: 910-861-4999

†Component Specialties, Inc.
8585 Commerce Park Drive, suite 590
Houston 77036
Tel: (713) 771-7237
TWX: 910-881-2422

Hamilton/Avnet Electronics
2401 Rutland
Austin 78758
Tel: (512) 837-8911

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 661-4111
TWX: 910-860-5371

†Hamilton/Avnet Electronics
3939 Ann Arbor Drive
Houston 77063
Tel: (713) 780-1771

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800

WASHINGTON

†Almac/Stroum Electronics
5811 Sixth Ave. South
Seattle 98108
Tel: (206) 763-2300
TWX: 910-444-2067

Arrow Electronics, Inc.
Electronics Distribution Division
1059 Andover Park East
Tukwila 98188
Tel: (206) 575-0907

†Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5844

WASHINGTON (cont.)

†Wyle Distribution Group
1750 132nd Avenue NE
Bellevue 98005
Tel: (206) 453-8300
TWX: 910-443-2526

WISCONSIN

†Arrow Electronics
430 W. Rawson Avenue
Oak Creek 53154
Tel: (414) 764-6600
TWX: 910-338-0026

†Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

†L.A. Varah Ltd.
4742 14th Street N.E.
Calgary T2D 6L7
Tel: (403) 230-1235
TWX: 018-258-97

Zentronics
9224 27th Avenue
Edmonton T6N 1B2
Tel: (403) 463-3014

Zentronics
3651 21st N.E.
Calgary T2E 6T5
Tel: (403) 230-1422

BRITISH COLUMBIA

†L.A. Varah Ltd.
2077 Alberta Street
Vancouver V5Y 1C4
Tel: (604) 873-3211
TWX: 610-929-1068

Zentronics
550 Cambie St.
Vancouver V6B 2N7
Tel: (604) 688-2533
TWX: 04-5077-89

MANITOBA

L.A. Varah
1-1832 King Edward Street
Winnipeg R2R 0N1
Tel: (204) 633-6190
TWX: 07-55-365

Zentronics
590 Berry St.
Winnipeg R3H 0S1
Tel: (204) 775-8661

ONTARIO

†Hamilton/Avnet Electronics
3688 Rexwood Road, Units G & H
Mississauga L4V 1M5
Tel: (416) 677-7432
TWX: 610-492-8860

†Hamilton/Avnet Electronics
1735 Courtwood Crescent
Ottawa K2C 3J2
Tel: (613) 226-1700
TWX: 053-4971

†L.A. Varah, Ltd.
505 Kenora Avenue
Hamilton L8E 3P2
Tel: (416) 561-9311
TWX: 061-8349

†Zentronics
141 Catherine Street
Ottawa K2P 1C3
Tel: (613) 238-6411
TWX: 053-3636

†Zentronics
1355 Meyerside Drive
Mississauga, Ontario L5T 1C9
Tel: (416) 676-9000
Telex: 06-983-657

QUEBEC

†Hamilton/Avnet Electronics
2670 Sabourin Street
St. Laurent H4S 1M2
Tel: (514) 331-6443
TWX: 610-421-3731

Zentronics
5010 Pare Street
Montreal H4P 1P3
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

intel
3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

INTERNATIONAL SALES AND MARKETING OFFICES

September 1980

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Micro Sistemas S.A.
9 De Julio 561
Cordoba
Tel: 54-51-32-880
TELEX: 51837 BICCO

AUSTRALIA

A.J.F. Systems & Components Pty. Ltd.
310 Queen Street
Melbourne
Victoria 3000
Tel:
TELEX:

Warburton Franki
Corporate Headquarters
372 Eastern Valley Way
Chatswood, New South Wales 2067
Tel: 407-3261
TELEX: AA 21299

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotenmulgasse 26
A 1120 Vienna
Tel: (0222) 83 63 96
TELEX: (01) 1532

Rekirsch Elektronik Geraete GmbH
Lichtensteinstrasse 97
A1000 Vienna
Tel: (222) 347646
TELEX: 74759

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

BRAZIL

Icotron S.A.
0511-Av. Mutinga 3650
6 Andar
Pirituba-Sao Paulo
Tel: 261-0211
TELEX: (011) 222 ICO BR

CHILE

DIN
Av. Vic. Mc kenna 204
Casilla 6055
Santiago
Tel: 227 564
TELEX: 3520003

CHINA

C.M. Technologies
525 University Avenue
Suite A-40
Palo Alto, CA 94301

COLOMBIA

International Computer Machines
Carrera 7 No. 72-34
Apdo-Aereo 19403
Bogota 1
Tel: 211-7282
TELEX: 71314 INCO

CYPRUS

Cyprus Eltrom Electronics
P.O. Box 5393
Nicosia
Tel: 21-27982

DENMARK

STL-Lyngso Komponent A/S
Ostmarken 4
DK-2860 Soborg
Tel: (01) 67 00 77
TELEX: 22990

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen
Tel: (01) 83 50 90
TELEX: 19037

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel: 0-692 6022
TELEX: 124 224 Ftron SF

FRANCE

Celdis S.A.*
53, Rue Charles Frerot
F-94250 Gentilly
Tel: (1) 581 00 20
TELEX: 200 485

Feutrier
Rue des Trois Glorieuses
F-42270 St. Priest-en-Jarez
Tel: (77) 74 67 33
TELEX: 300 0 21

Metrologie*
La Tour d'Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel: 791 44 44
TELEX: 611 448

Tekelec Airtronic*
Cite des Bruyeres
Rue Carle Vernet
F-92310 Sevres
Tel: (1) 534 75 35
TELEX: 204552

GERMANY

Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Munich 80
Tel: (089) 434061
TELEX: 522561

Jermyn GmbH
Postfach 1180
D-6077 Camberg
Tel: (06434) 231
TELEX: 484426

Kontron Elektronik GmbH
Breslauerstrasse 2
8057 Eching B
D-8000 Munich
Tel: (89) 319.011
TELEX: 522122

Neye Enatechnik GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel: (04106) 6121
TELEX: 02-13590

GREECE

American Technical Enterprises
P.O. Box 156
Athens
Tel: 30-1-8811271
30-1-8219470

HONG KONG

Schmidt & Co.
28/F Wing on Center
Connaught Road
Hong Kong
Tel: 5-455-644
TELEX: 74766 Schmc Hx

INDIA

Micronic Devices
104/109C, Nirmal Industrial Estate
Sion (E)
Bombay 400022, India
Tel: 486-170
TELEX: 011-5947 MDEV IN

ISRAEL

Eastronics Ltd.*
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: 475151
TELEX: 33638

ITALY

Eledra 3S S.P.A.*
Viale Elvezia, 18
I 20154 Milan
Tel: (02) 34.93.041-31.85.441
TELEX: 332332

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. Room 407
2-14-1 Asano, Kokura
Kita-Ku, Kitokyushu City 802
Tel: (093) 511-6471
TELEX: AECKY 7126-16

Hamilton-Avnet Electronics Japan Ltd.
YU and YOU Bldg. 1-4 Horidome-Cho
Nihonbashi
Tel: (03) 662-9911
TELEX: 2523774

Ryoyo Electric Corp.
Konwa Bldg.
1-12-22, Tsukiji, 1-Chome
Chuo-Ku, Tokyo 104
Tel: (03) 543-7711

Tokyo Electron Ltd.
No. 1 Higashikata-Machi
Midori-Ku, Yokohama 226
Tel: (045) 471-8811
TELEX: 781-4473

KOREA

Koram Digital
Room 909 Woonam Bldg.
7, 1-KA Bongre-Dong
Chung-Ku Seoul
Tel: 23-8123
TELEX: K23542 HANSINT

Leewood International, Inc.
C.P.O. Box 4046
112-25, Sokong-Dong
Chung-Ku, Seoul 100
Tel: 28-5927
CABLE: "LEEWOOD" Seoul

MEXICO

Proveedora Electronica, S.A. (Proesa)
Prol. Moctezuma Ote. 24
Col. Romero de Terreros
Apdo. Postal 21-139
Mexico 21, D.F.
TELEX: 017-72402 SAULME

NETHERLANDS

Inelco Nether. Comp. Sys. BV
Turfstekerstraat 63
Aalsmeer 1431 D
Tel: (2977) 28855
TELEX: 14693

Koning & Hartman
Koperwerf 30
2544 EN Den Haag
Tel: (70) 210.101
TELEX: 31528

NEW ZEALAND

W. K. McLean Ltd.
P.O. Box 18-065
Glenn Innes, Auckland, 6
Tel: 587-037
TELEX: NZ2763 KOSFY

NORWAY

Nordisk Elektronik (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: 02 78 62 10
TELEX: 17546

PORTUGAL

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
Lisboa 1
Tel: (19) 545313
TELEX: 14347 GESPIC

SINGAPORE

General Engineers Associates
Blk 3, 1003-1008, 10th Floor
P.S.A. Multi-Storey Complex
Telok Blangah/Pasir Panjang
Singapore 5
Tel: 271-3163
TELEX: RS23987 GENERCO

SOUTH AFRICA

Electronic Building Elements
Pine Square
18th Street
Hazelwood, Pretoria 0001
Tel: 789 221
TELEX: 30181SA

SPAIN

Interface S.A.
Ronda San Pedro 22, 3°
Barcelona 10
Tel: 301 78 51
TWX: 51508

ITT SESA
Miguel Angel 16
Madrid 10
Tel: (1) 4190957
TELEX: 27707/27461

SWEDEN

AB Gosta Backstrom
Box 12009
10221 Stockholm
Tel: (08) 541 080
TELEX: 10135

Nordisk Electronik AB
Box 27301
S-10254 Stockholm
Tel: (08) 635040
TELEX: 10547

SWITZERLAND

Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel: (01) 60 22 30
TELEX: 56788

TAIWAN

Taiwan Automation Co.*
3d Floor #75, Section 4
Nanking East Road
Taipei
Tel: 771-0940
TELEX: 11942 TAIAUTO

TURKEY

Turkelek Electronics
Apapurk Boulevard 169
Ankara
Tel: 189483

UNITED KINGDOM

Comway Microsystems Ltd.
Market Street
68-Bracknell, Berkshire
Tel: (344) 51654
TELEX: 847201

G.E.C. Semiconductors Ltd.
East Lane
North Wembley
Middlesex HA9 7PP
Tel: (01) 904-9303/908-4111
TELEX: 28817

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 501.44
TELEX: 95142

Rapid Recall, Ltd.
6 Soho Mills Ind. Park
Wooburn Green
Bucks, England
Tel: (6285) 24961
TELEX: 849439

Sintrom Electronics Ltd.*
Arkwright Road 2
Reading, Berkshire RG2 0LS
Tel: (0734) 85464
TELEX: 847395

VENEZUELA

Componentes y Circuitos
Electronicos TTLCA C.A.
Apartado 3223
Caracas 101
Tel: 718-100
TELEX: 21795 TELETIPOS

*Field Application Location

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

INTERNATIONAL SALES AND MARKETING OFFICES

September 1980

INTEL® MARKETING OFFICES

AUSTRALIA

Intel Semiconductor Pty., Ltd.
Suite 2, Level 15, North Point
100 Miller Street
North Sydney, NSW, 2060
Tel: 450-847
TELEX: AA 20097

BELGIUM

Intel Corporation S.A.
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02) 660 30 10
TELEX: 24814

DENMARK

Intel Denmark A/S*
Lyngbyvej 32 2nd Floor
DK-2100 Copenhagen East
Tel: (01) 18 20 00
TELEX: 19567

FINLAND

Intel Finland OY
Sentnerikuja 3
SF - 00400 Helsinki 40
Tel: (0) 558531
TELEX: 123 332

FRANCE

Intel Corporation, S.A.R.L.*
5 Place de la Balance
Silic 223
94528 Rungis Cedex
Tel: (01) 687 22 21
TELEX: 270475

GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
8000 Muenchen 2
Tel: (089) 53 891
TELEX: 523 177

Intel Semiconductor GmbH
Mainzer Strasse 75
6200 Wiesbaden 1
Tel: (06121) 700874
TELEX: 04186183

Intel Semiconductor GmbH
Wernerstrasse 67
P.O. Box 1460
7012 Fellbach
Tel: (0711) 580082
TELEX: 7254826

Intel Semiconductor GmbH
Hohenzollern Strasse 5
3000 Hannover 1
Tel: (0511) 327081
TELEX: 923625

Intel Semiconductor GmbH
Oberrathstrasse 2
4000 Duesseldorf 30
Tel: (0211) 651054-6
TELEX: 8586977

HONG KONG

Intel Semiconductor Ltd.
99-105 Des Voeux Rd., Central
18F, Unit B
Hong Kong
Tel: 5-450-847
TELEX: 63869

ISRAEL

Intel Semiconductor Ltd.*
P.O. Box 2404
Haifa
Tel: 972/452 4261
TELEX: 92246511

ITALY

Intel Corporation Italia, S.p.A.
Corso Sempione 39
I-20145 Milano
Tel: 2/34.93287
TELEX: 311271

JAPAN

Intel Japan K.K.*
Flower Hill-Shinmachi East Bldg.
1-23-9, Shinmachi, Setagaya-ku
Tokyo 154
Tel: (03) 426-9261
TELEX: 781-28426

NETHERLANDS

Intel Semiconductor B.V.
Cometongebouw
Westblaak 106
3012 Km Rotterdam
Tel: (10) 149122
TELEX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

SWEDEN

Intel Sweden A.B.*
Box 20092
Alpvagen 17
S-16120 Bromma
Tel: (08) 98 53 90
TELEX: 12261

SWITZERLAND

Intel Semiconductor A.G.
Forchstrasse 95
CH 8032 Zurich
Tel: 1-55 45 02
TELEX: 557 89 ich ch

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
5 Hospital Street
Nantwich, Cheshire CW5 5RE
Tel: (0270) 62 65 60
TELEX: 36620

Intel Corporation (U.K.) Ltd.
Dorcan House
Eldine Drive
Swindon, Wiltshire SN3 3TU
Tel: (0793) 26101
TELEX: 444447 INT SWN

*Field Application Location

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A./T-45/1080/10K CBM DLR SW