SIGNETICS 8X300 REFERENCE MANUAL

$3.50

SIGNETICS 8X300 REFERENCE MANUAL

signetics

PREFACE

The 8X300 Reference Manual was compiled from various sources to assist designers in the use and evaluation of the 8X300 Microcontroller. Included are basic descriptions of the microcontroller, support circuits, development aids and applications.

Additional assistance in the use of the 8X300 can be obtained from personnel in Signetics Field Sales Offices and the documents listed below:

- 8X300 Programming Course
- Signetics Bipolar/MOS Microprocessor Data Manual
- Signetics Bipolar & MOS Memory Data Manual
- The Microcontroller Cross Assembler Program (MCCAP) User's Guide
- Designing a Floppy Disc Controller an Applications Manual
- Microprocessor Evaluation Guide

Copies of this manual and those referenced above are available from Signetics Field Sales Offices.

TABLE OF CONTENTS

PRODUCT DESCRIPTION

ONE-CHIP BIPOLAR MICROCONTROLLER APPROACHES BIT-SLICE PERFORMANCE

8X300 combines best of both worlds with 250-nanosecond cycle time and powerful instruction set

by John Nemec, *Signetics Corp., Sunnyvale, Calif.*

□ While refinements in the metal-oxide-semiconductor process have improved the performance of 8-bit MOS microprocessors, the need for really high speed still drives designers to the bipolar bit-slice machines. Advances in bipolar Schottky technology, however, now allow the design of a single-chip microprocessor with the computational ability and the performance advantages that approach those found in bit-slice designs.

The 8X300 microprocessor from Signetics Corp. is the first such device. Not only does it have a cycle time of 250 nanoseconds, but it also has an improved architecture that makes it extremely efficient; hence its throughput is vastly superior to those of MOS devices. The interfacing bus structure, for example, is partitioned into two banks, and in a single instruction cycle the processor can accept data from a port on one bank, operate on it, and deposit the result in a port on the other bank. Since instruction fetching is concurrent with data operations, and both are executed in one 250-ns instruction cycle, the 8X300 is as fast as bipolar bit-slice systems on a microcycle basis.

Moreover, the 8X300 is easier to program than bit-slice devices. The powerful instructions have simple mnemonic representations of the functions they perform, such as ADD for the addition function, and these mnemonics can be directly translated into their octal representation. With these conveniences, several hundred lines of program code can readily be written by hand. Consequently, for tasks of less than 500 instructions, no assembler is needed—only a simple conversion is required to generate the actual program-memory content.

Its ease of programming and flexible interfacing structure make the 8X300 a natural as a subsystem and peripheral-device controller—one requiring little additional hardware for such applications. And the speed of the 8X300 allows it to handle control functions that MOS processors cannot, such as direct-memory-access interfacing, for example.

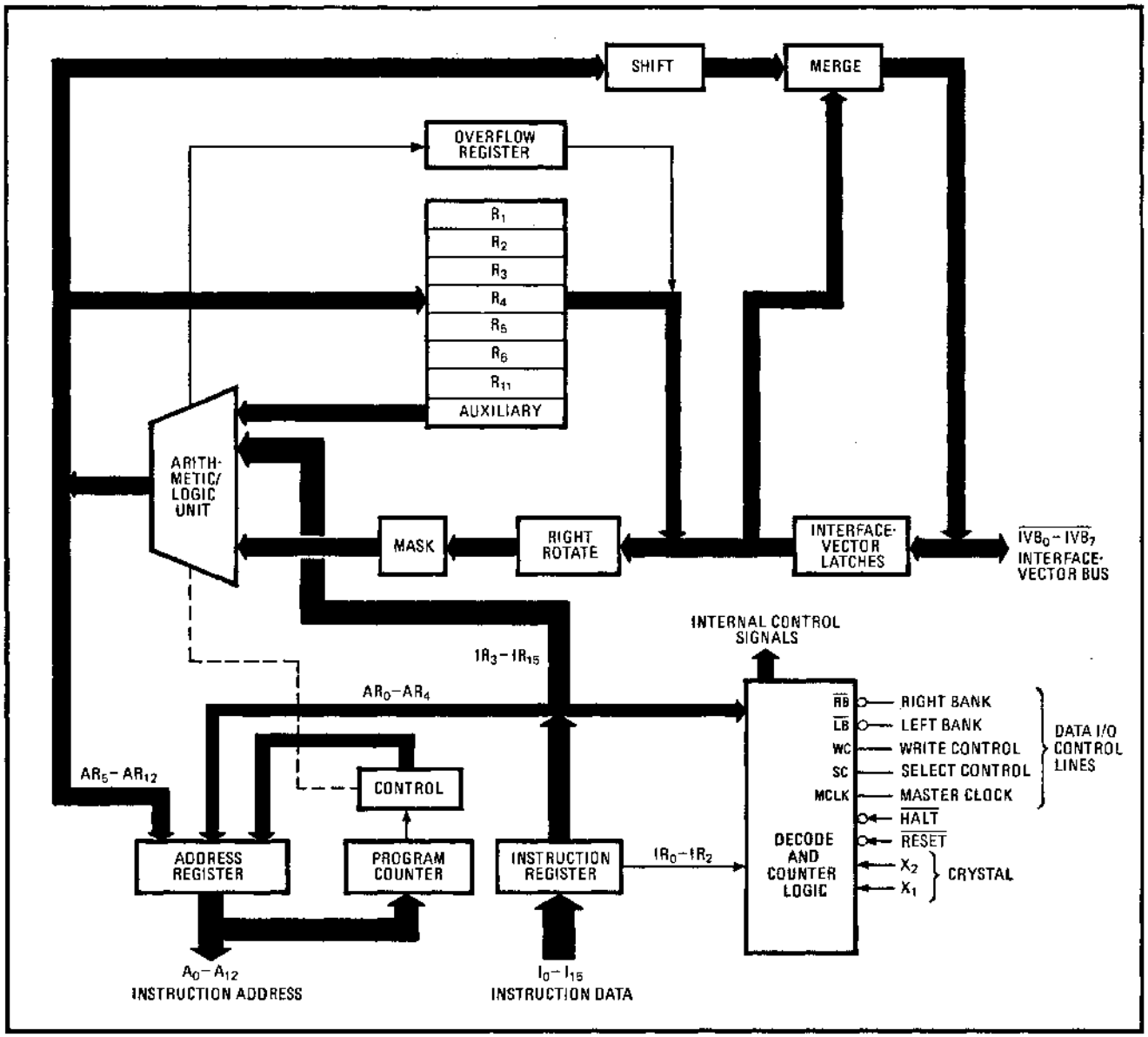
Novel architecture

The architecture of the 8X300 processor is shown in Fig. 1. The chip includes full instruction-decoding logic that interprets the particular class of instructions, such as input/output or arithmetic and logic, and performs the indicated operation. The decoding and control logic supplies all internal signals for the processor, as well as signals on the control lines for directing the data input and output.

The processor also contains its own program counter, which is automatically incremented upon execution of the instruction. The counter may also be left unchanged or loaded with a new value. Control of the current

*NOTE

This article originally appeared in ELECTRONICS Magazine September 1, 1977. It is reprinted here through the courtesy of and with permission of ELECTRONICS Magazine.



1. High-performance microcontroller. The Signetics 8X300 processor, a bipolar Schottky device with a 250-ns cycle time, is capable of processing at throughputs rivaling those of bit-slice machines. Fetching of its 16-bit instructions is concurrent with data operations; and with a bus partitioned into right and left banks, the 8X300 can, in a single instruction cycle, accept data, operate on it, and deposit the result.

address is provided by the address register and may be derived completely or partially from the program counter, from the instruction data lines (AR_0 through AR_4), or from the output of the arithmetic/logic unit (lines AR_5 through AR_{12}). Because of this flexible instruction-address scheme, the order of execution may be altered by instructions or under conditions determined from selected data.

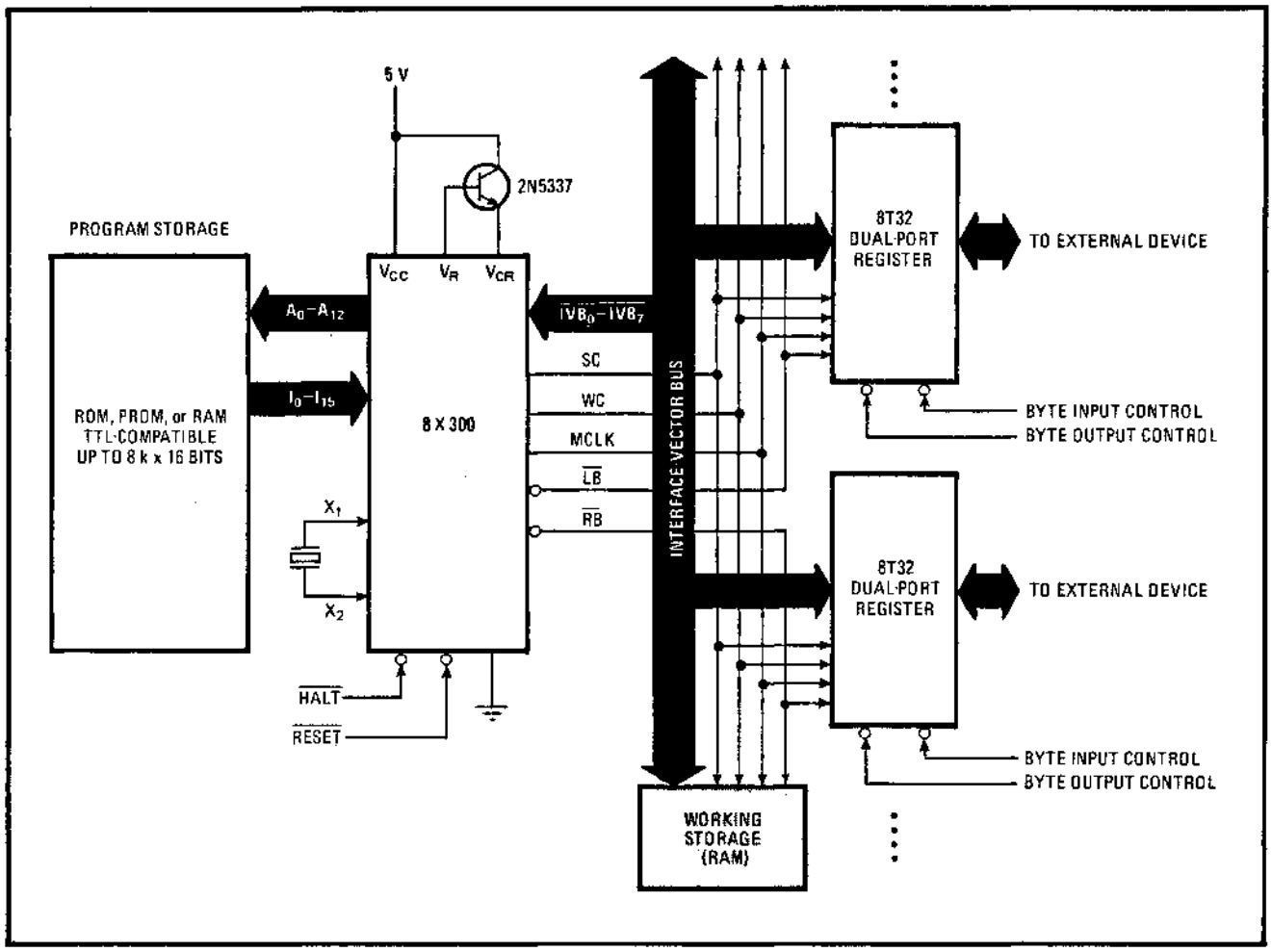
The read/write registers

The processor manipulates 8-bit data bytes. Internal data is stored in eight 8-bit read/write registers—R_1 through R_6, R_{11}, and an auxiliary register. The auxiliary register holds one of the operands used in two-operand instructions, such as ADD or AND, and a single-bit overflow register stores the carry-over bit from additional operations.

Further, the addition of the rotate, mask, shift, and merge functions to the ALU extends the 8X300's operating capability and enhances its efficiency.

Interfacing with external circuitry is through an 8-bit bus called the interface-vector bus and consisting of lines IV_0 through IV_7. The bus carries both address and data information, and the accompanying data-I/O control lines tell the external circuitry which of the two types of information is on the bus. These lines include write- and select-control, right- and left-bank-signal, and master-clock lines.

Since the interface-vector bus carries addresses as well as data, I/O ports on the external circuits must be enabled before data transfer can take place. This is usually accomplished by placing an address on the bus under program control and then activating the select-control line, which indicates that a valid address is on the bus. When presented with an address, each of the possible 512 I/O ports either enables itself upon



2. System design. A typical configuration has memory for program storage and latches for up to 512 directly addressed I/O ports, which are divided among left and right banks. All addresses and data are carried by the interface-vector bus and directed by control lines.

identifying the address as its own or disables itself if the addresses do not match.

The processor can directly address all of the I/O ports without the need for a decoder. The bus, and the interface-vector bytes on it, are bidirectional.

Within the processor, the interface-vector bytes are addressed in a unique fashion. Each byte has an 8-bit field-programmable address. When a given address is selected, the byte is automatically designated, and the 8X300 can then communicate with the I/O device. Moreover, once enabled, the addresses remain so until the processor changes them. This direct-addressing feature is especially convenient if a few ports are to be accessed frequently. However, if the time required for this operation is an imposition on the user, instruction memory can be extended so that the selection of ports is automatic upon instruction fetch.

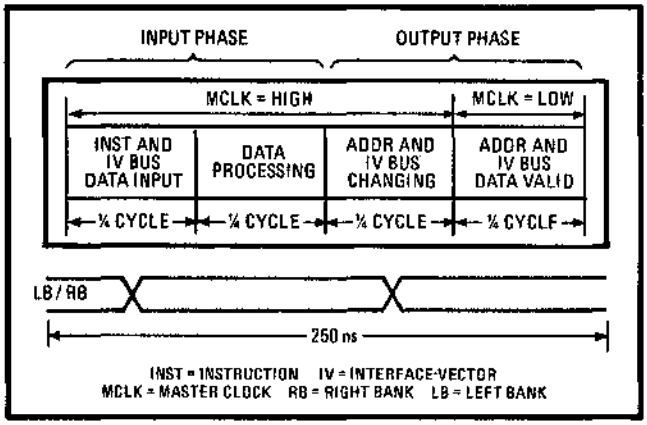
In extending the memory, an extra field is appended to each instruction through an additional bus applied to each I/O port. The address field may be as wide as required to serve all system I/O ports and, if necessary, to serve memory. The address field contains two addresses, since the interface-vector bus is partitioned into right and left banks.

Partitioning of the interface-vector bus into two banks allows the 8X300 to select ports dynamically. The processor uses the left-bank (LB) and right-bank (RB) data-control lines as master enables for the I/O ports, as shown in the typical interconnect scheme of Fig. 2. Any two I/O ports can be active at the same time provided they are on opposite banks, and the ports recognize address, data, and controls only when enabled by the bank signal to which each is connected. Bank partitioning can thus be considered a ninth address bit that is alterable by the processor within an instruction, and it is this additional bit that permits direct addressing of 512, or 2^9, I/O ports.

In a general data operation between two I/O ports, first an address is presented to one bank that enables an I/O port and disables all others on the bank. Next, another address is presented to the opposite bank, effecting a similar selection there. Then the operation between the two takes place.

Instruction cycle

Each 8X300 operation is executed in one instruction cycle, which is subdivided into four quarter cycles as shown in Fig. 3. The instruction address for an operation is presented at the output of the processor during the third quarter of the previous instruction cycle. If the system memory is fast enough, the instruction returns to the processor during the first quarter of the cycle in



3. Instruction cycle. Two I/O ports may be dynamically selected in a single cycle if they are on opposite banks. Complementary LB and RB control signals change state during first and third quarter-cycles to accept data from one port and deposit it in the other.

which it is to be executed. The decoded instruction then directs the operation of the processor throughout the cycle.

In terms of processing data, the instruction cycle may be viewed as having two halves, an input and an output phase. During the first half of the instruction cycle, data is brought into the processor and stored in an interface-vector latch. Storage is completed during the first quarter cycle, and in the next quarter cycle the data is processed through the ALU. In the second half cycle, the data is presented to the bus and finally clocked into the designated I/O port.

Bank selection during the input and output phases is independent. Thus data may be received from the right bank, processed, and then deposited in the left bank or vice versa, or may even be sent to and from the same bank. Bank selection during instruction cycles is specified by the instruction.

Much of the strength of the 8X300 architecture lies in the powerful instruction set that controls the processor. The instruction words, each 16 bits in length, are made up of several fields that include the operational code, the source and destination fields, and the length field.

Instruction fields

The contents of each field can be represented by a set of octal digits, to simplify the task of coding. These digits have a direct relation to the specific operations that the data undergoes in its travel along the 8X300's internal data paths. The op code for addition, for example, is 01. In an operation between two I/O ports, the first of the source field's two digits specifies the bank and the second prescribes the number of bits to be rotated to the right. Similarly, the first digit of the destination field again specifies a bank, while the second digit prescribes the number of bits to be shifted. The length field specifies the number of bits to be accepted for operation in the ALU.

The capabilities of the 8X300 are such that it can, with few additional devices, perform all the functions required of an intelligent terminal. Whereas a typical system of this kind is constructed from many small- or medium-scale-integrated circuits for the control portion of the terminal—cathode-ray-tube display, dot-matrix graphics, keyboard data entry, and host or mass-storage interface—and a microprocessor for the intelligence portion—data manipulation and number crunching—the 8X300 performs most of both the control and intelligence functions. The result is a system with fewer parts and far more capabilities for the money.

Intelligent terminal

The 8X300-based intelligent terminal depicted in Fig. 4 consists of four major sections. First is the interface section, which implements the standard RS-232C interface with the host or main-data storage. The Signetics 2651 programmable communications interface performs the serial-parallel data interconversion for byte processing by the 8X300 and for display-refresh memory. The interface section includes a keyboard that is scanned by the processor for operator inputs.

The second major section comprises the video circuitry that generates all pixel, character, and line timing, as well as the composite video signal. This circuitry also supplies signals to the refresh-memory address counter for proper timing of character codes and attributes—blinking, reverse video, and so on. Synchronization, blanking, video, and attribute signals are then summed in the video mixer to produce the signal for driving standard monitors.

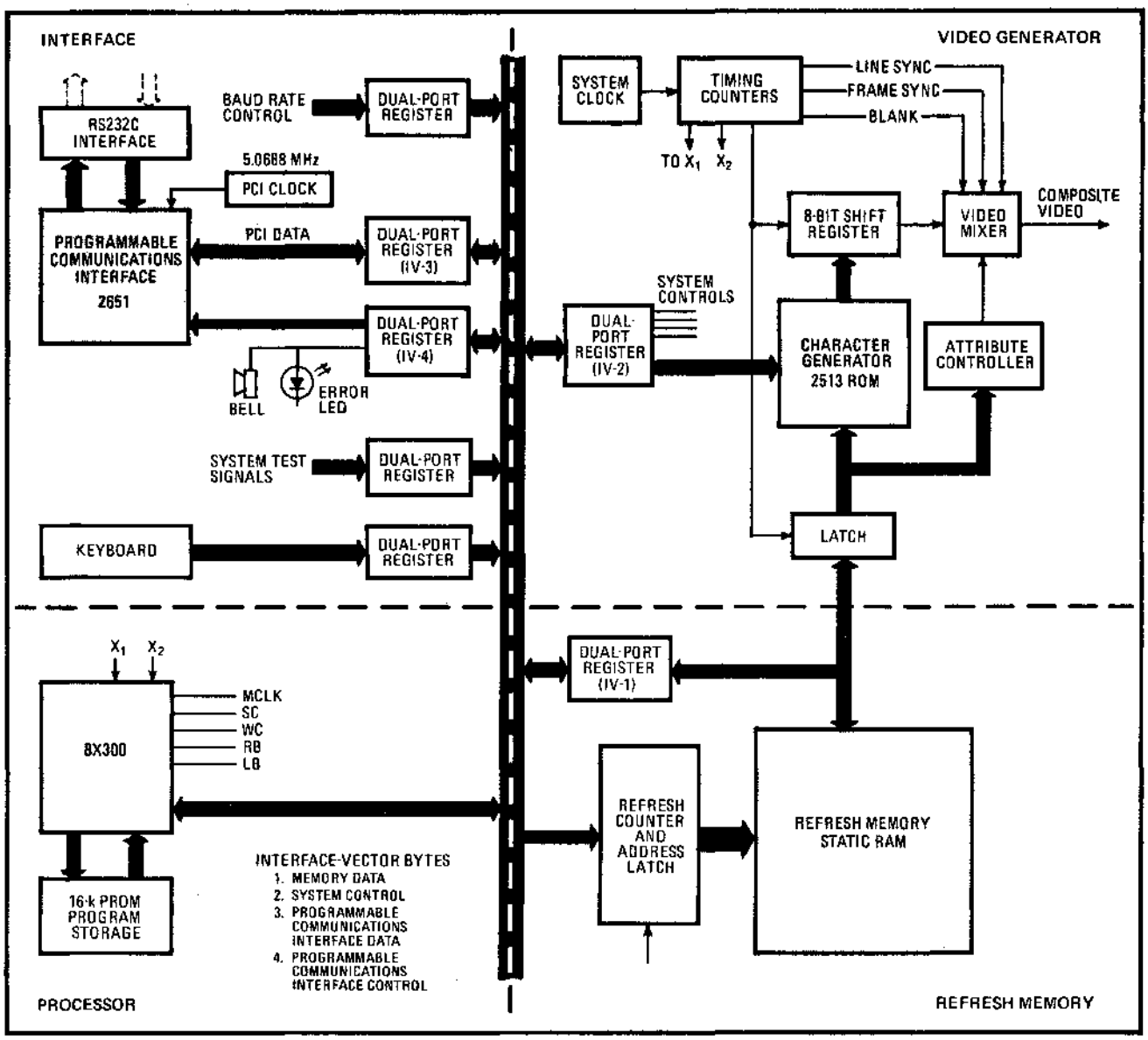
The third section, the refresh memory, stores the characters to be displayed on the screen. Its characters are addressed under control of the video-generator circuitry. The circuitry also provides for timing and control to facilitate interleaved access to the memory by the 8X300 processor. Interleaving permits the processor to examine or modify the refresh memory without disturbing the on-screen display or the video-refresh process.

The processor's role

The final section is the processor, the terminal's intelligence center. It performs both the executive control—managing the other three sections—and the data-processing function required by the terminal.

Since the 8X300 controls the refresh memory and video-generator sections and thus provides the timing for each frame, all frame-related features, such as scrolling and command-scanning, can be easily implemented. But most important, the processor also performs all the data manipulations—the intelligence within the terminal. The processor may accept input data from the host or the keyboard and can then present the data in proper format by means of the CRT display. Information may be entered or modified by the user and then returned to the host after the essential data items have been extracted from the refresh-memory file. This process, tailored to the user's needs, may be accomplished fully by the 8X300 microprocessor in firmware.

The basic control program, including keyboard scan, RS-232C interface service, edit, scroll, cursor, and so forth, requires only 502 words of program storage. Thus the entire terminal can be coded with two 4,096-bit programmable read-only memories. The balance of parts for the entire terminal hardware include the 2651



4. Intelligent terminal. The 8X300 builds an intelligent terminal that, with a total of about only 40 devices, is low in cost while high in capabilities. Included in the terminal are RS-232C interfacing, keyboard scanning, character generation with attributes, and fully interleaved refresh memory. The processor manages the other three sections and adds intelligence in data handling and number crunching.

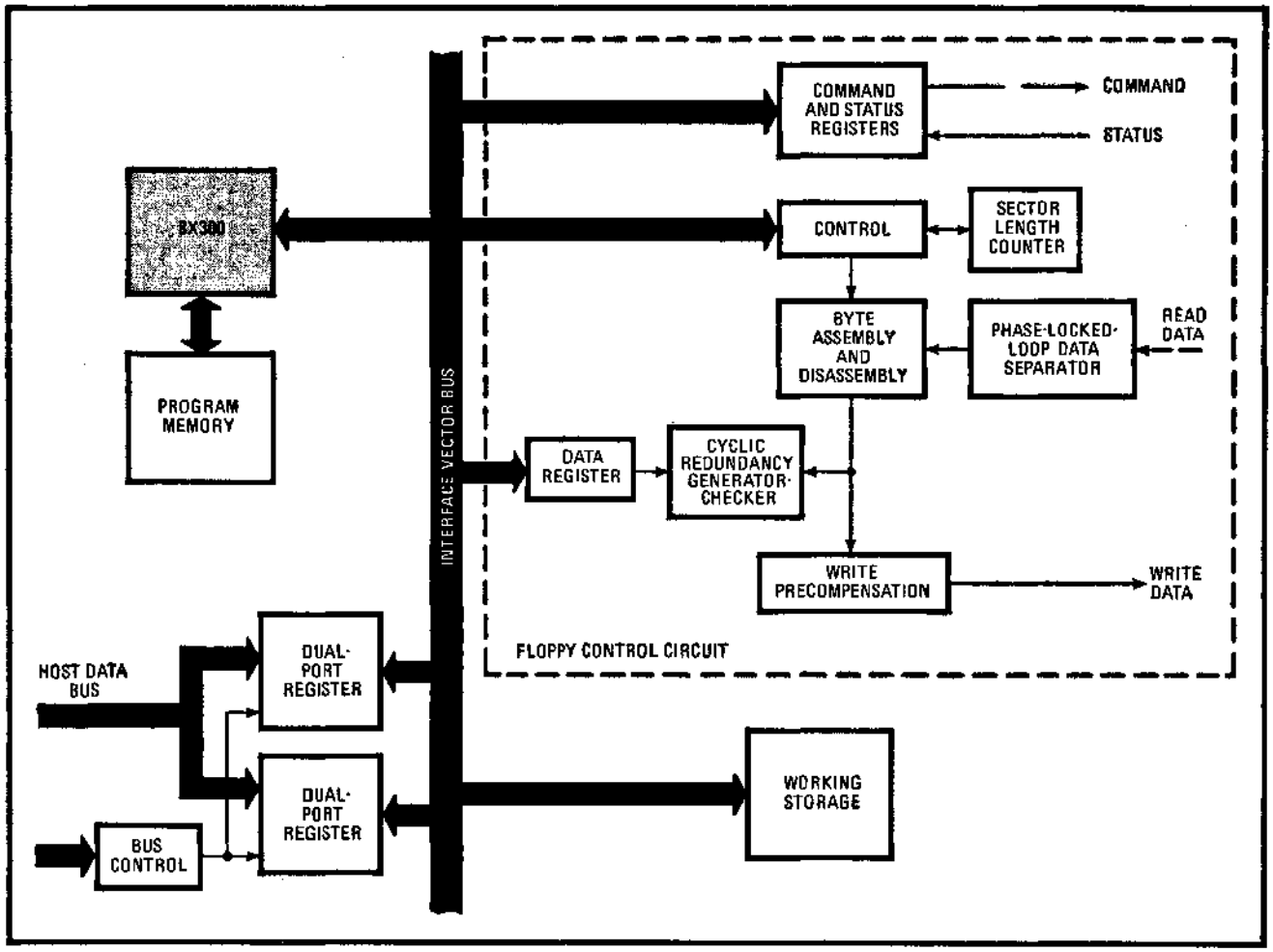
programmable interface chip, the random-access memory for display refresh, a field-programmable logic array, and several low-power Schottky components, in addition to the 8X300. The use of about only 40 medium- and large-scale-integrated parts to construct the complete system contrasts with the over 110 small- and medium-scale-integrated components needed at present to build dumb terminals.

Floppy controller

The 8X300 can be used in a very practical floppy-disk controller circuit that has few parts and offers highly reliable control and data transfer and complete interfacing with the host processor. The processor's high speed enables it to handle any density—single, dual, and quad formats. With only minor firmware changes, a single hardware configuration can accommodate all popular disk drives and a variety of host processors. What is more, it can do so within a single operating system—all that is needed is to switch from one set of firmware to another.

The floppy-disk controller, shown in Fig. 5, has basically two major functional parts, plus minor support functions. The two major sections are the processor and the control circuit.

The 8X300 communicates with the surrounding circuits through the interface-vector bus and controls the disk drives through command and status registers. Operating efficiency is high because selected control and status bits share the same bidirectional I/O register, allowing both polling (status monitoring) and command to take place in as short a time as 250 ns. The processor's high speed allows simultaneous control of several disk drives for operations such as overlap seeks, where one drive is accessing while another reads or writes data. The 8X300's limits are reached only during tight-loop data



5. Floppy-disk controller. High performance of the 8X300 allows it to effectively control a floppy-disk drive. The processor not only handles track and sector addressing but actually oversees data transfer. It also provides interfacing with the working store or directly with the host.

transfers that have to be extremely fast, but since these periods are very short, they present no significant system delays.

The 8X300 builds reliability into the disk operations by overseeing the actual data transfer, in addition to positioning the read/write head. The intelligence of the processor can be used to reread data after it has been written or even to mask out bad tracks that fail to read correctly after, say, 10 rewrites.

Efficient operation

Registers within the 8X300 monitor the track and sector addresses, making seeks and sector accesses highly efficient. In setting up the data transfers, the 8X300 monitors both interleaved data on the disk and the system clock for address-mark detection; it then examines and moves data a byte at a time once the controller circuit has been synchronized with the byte boundaries. Under control of the processor, disk data may be transferred to working storage (which can interface directly with the processor bus) for subsequent transfer to the host or, if it can be accommodated, transferred immediately to the host.

Very little external logic is needed for direct-memory-access interfacing between the 8X300 and the host. Transfer of data, commands, and status takes place through eight bidirectional I/O ports that are compatible with transistor-transistor logic and are supplemented with additional logic for direct connection to the host. The protocol is established in processor microprogramming, which may be conveniently altered for a variety of host types. Total program size for disk control and interfacing the 8X300 with the host processor is less than 800 instructions.

The other major section, the control circuit, has the hardware, which does a better job of recovering data from the disk than microprocessor software could. Functions include phase-locked-loop data separation, cyclic redundancy generating and checking, data-byte assembly and disassembly, and precompensation for widening the write pulses applied to the disk. These circuit functions, under control of the 8X300, can easily be instructed to operate for either single or dual recording densities, and the system can be expanded to handle most types of quad-density encoding.

Counting the control circuit as two LSI devices, the entire floppy-disk controller can be built from about 12 components. The system is complete in that it not only controls the access of disk data, but also provides the interface between any host and any off-the-shelf floppy-disk drive, and monitors and checks data transfer for ultimate reliability. □

DESCRIPTION

The Signetics 8X300 Microcontroller is a monolithic, high-speed microprocessor implemented with bipolar Schottky technology. As the central processing unit, CPU, it allows 16-bit instructions to be fetched, decoded and executed in 250ns. A 250ns instruction cycle requires maximum memory access of 65ns, and maximum I/O device access of 35ns.

Microcontroller instructions operate on 8-bit, parallel data. Logic is distributed along the data path within the Microcontroller. Input data can be rotated and masked before being subject to an arithmetic or logical operation; and output data can be shifted and merged with the input data, before being output to external logic. This allows 1- to 8-bit I/O and data memory fields to be accessed and processed in a single instruction cycle.

PROGRAM STORAGE INTERFACE

Program Storage is typically connected to the A0-A12 (A12 is least significant bit) and I0-I15 signal lines. An address output on A0-A12 identifies one 16-bit instruction word in program storage. The instruction word is subsequently input on I0-I15 and defines the Microcontroller operations which are to follow.

The Signetics 82S115 PROM, or any TTL compatible memory, may be used for program storage.

I/O DEVICES INTERFACE

An 8-bit I/O bus, called the Interface Vector (IV) data bus, is used by the Microcontroller to communicate with 2 fields of I/O devices. The complementary $\overline{LB}$ and $\overline{RB}$ signals identify which field of the I/O devices is selected.

Both I/O data and I/O address information can be output on the IV bus. The SC and WC signals are typically used to distinguish between I/O data and I/O address information as follows:

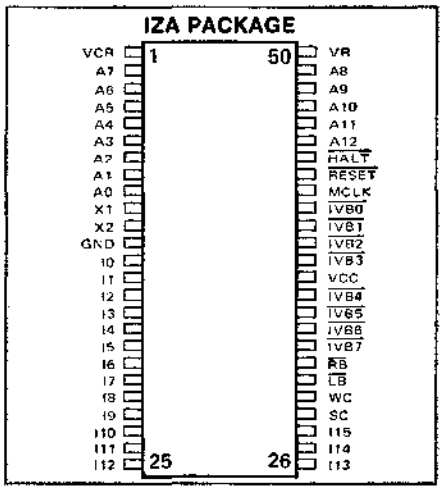
SC	WC	
1	0	I/O address is being output on IV bus
0	1	I/O data is being output on IV bus
0	0	I/O data is expected on the IV, bus, as input to the Microcontroller
1	1	Not generated by the Microcontroller

The Signetics 82SXXX series RAM, and the 8T32/33 may be attached to the IV bus.

FEATURES

- **185ns instruction decode and execute delay (with Signetics 8T32/33 I/O port)**
- **Eight 8-bit working registers**
- **Single instruction access to 1-bit, 2-bit, 3-bit or 8-bit field on I/O bus**
- **Separate instruction address, instruction, and I/O data buses**
- **On-chip oscillator**
- **Bipolar Schottky technology**
- **TTL inputs and outputs**
- **Tri-state output on I/O data bus**
- **+5 volt operation from 0° to 70°C**

PIN CONFIGURATION



PIN DESIGNATION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
2-9, 45-49	A0-A12:	Instruction address lines. A high level equals "1." These outputs directly address up to 8192 words of program storage. A12 is least significant bit.	Active high
13-28	I0-I15:	Instruction lines. A high level equals "1." Receives instructions from Program Storage. I_{15} is least significant bit.	Active high
33-36, 38-41	$\overline{IVB0}$-$\overline{IVB7}$	Interface Vector (IV) Bus. A low level equals "1." Bidirectional tri-state lines to communicate with I/O devices. $\overline{IVB7}$ is least significant bit.	Three-state Active low
42	MCLK:	Master Clock. Output to clock I/O devices, and/or provide synchronization for external logic	Active high
30	WC:	Write Command. High level output indicates data is being output on the IV Bus.	Active high
29	SC:	Select Command. High level output indicates that an address is being output on the IV Bus.	Active high
31	$\overline{LB}$:	Left Bank. Low level output to enable one of two sets of I/O devices ($\overline{LB}$ is the complement of RB).	Active low
32	$\overline{RB}$:	Right Bank. Low level output to enable one of two sets of I/O devices ($\overline{RB}$ is the complement of $\overline{LB}$).	Active low
44	$\overline{HALT}$:	Low level is input to stop the Microcontroller.	Active low
43	$\overline{RESET}$:	Low level is input to initialize the Microcontroller.	Active low
10-11	X1, X2:	Inputs for an external frequency determining crystal. May also be interfaced to logic or test equipment.	
50	VR	Reference voltage to pass transistor.	
1	VCR	Regulated output voltage from pass transistor.	
37	V_{CC}:	5V power connection.	
12	GND:	Ground.	

MICROCONTROLLER ARCHITECTURE

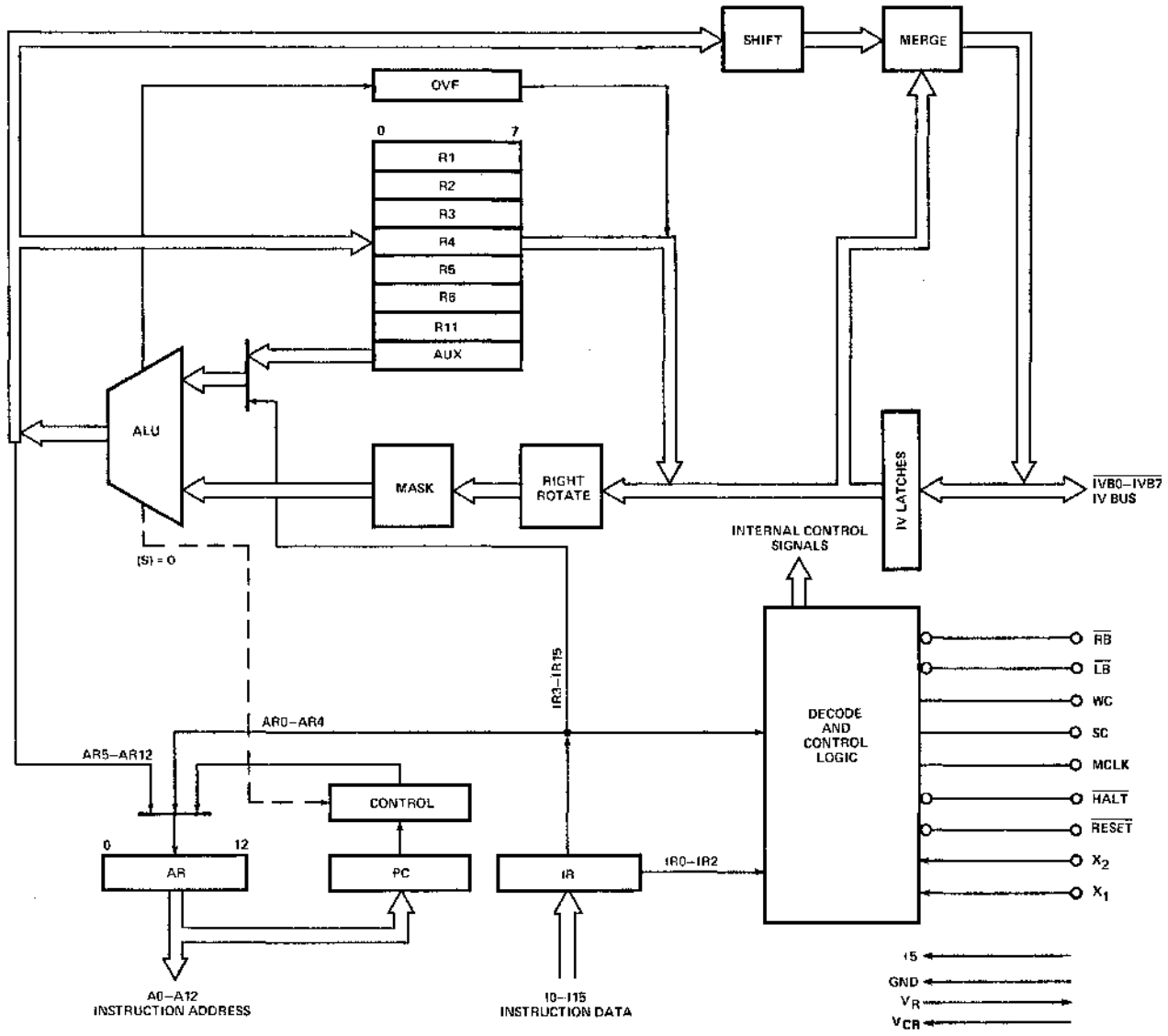


Figure 1

R1 — General working register
R2 — General working register
R3 — General working register
R4 — General working register
R5 — General working register
R6 — General working register
R11 — General working register
AUX — General working register. Contains second term for arithmetic or logical operations.

OVF — The least-significant bit of this register is used to reflect overflow status resulting from the most recent ADD operation (see Instruction Set Summary).

Instruction Register (IR)
— Holds the 16-bit instruction word currently being executed.

Program Counter (PC)
— Normally contains the address of the current instruction and is incremented to obtain the next instruction address.

Address Register (AR)
— A 13-bit register containing the address of the current instruction.

Table 1 INTERNAL REGISTERS

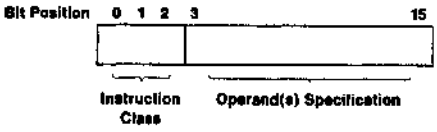
INSTRUCTION CYCLE

Each Microcontroller operation is executed in 1 instruction cycle, which may be as short as 250ns. The Microcontroller generates MCLK to synchronize external logic to the instruction cycle. Instruction cycles are subdivided into quarter cycles. MCLK is an output during the last quarter cycle.

During the third quarter cycle of an instruction, an address is output on A0-A12, identifying the location in program storage of the next instruction word. This instruction word defines the next instruction, which must be input on I0-I15 during the first quarter cycle of the next instruction cycle (see Table 2).

Instruction Set Summary

The 16-bit instruction word input on I0-I15 is decoded by the instruction decode logic to implement events that are to occur during the remainder of the instruction cycle. Generally the 16-bit instruction word is decoded as follows:



A detailed usage of the 13 "operand(s) specification" bits is given in following sections.

Three operation code bits allow for 8 instruction classes. The 8 instruction classes are summarized in Table 3. Each entry is referred to as an "instruction class" because the unique architecture of the Interpreter allows a number of powerful variations to be specified by the 13 operand(s) specification bits. A complete description of instruction formats and some instruction examples are provided in the Microprocessor Applications manual.

Data Processing

The Microcontroller architecture includes eight 8-bit working registers, an arithmetic logic unit (ALU), an overflow register, and the 8-bit IV Bus. Internal 8-bit data paths connect the registers and IV Bus to the ALU inputs, and the ALU output to the registers and IV Bus. Data processing logic is distributed along these internal 8-bit data paths. Rotate and mask logic precedes the ALU on the data entry path. Shift and merge logic follows the ALU on the data output path. All 4 sets of logic can operate on 8 data bits in a single instruction cycle (See Figure 1).

When less than 8 bits of data are specified for output to the IV bus by the ALU, the data field (shifted if necessary) is inserted into the prior contents of the IV bus latches. The IV bus latches contain data input at the start of an instruction. This data in the IV bus latches will be specified in the instruction as a) IV bus source data or b) data from an automatic read when the IV bus is specified as a destination. Therefore, IV bus bit positions outside an inserted bit field are unmodified.

Data Addressing

Sources and destinations of data are specified using a 5-bit octal number. The source and/or destination of data to be operated upon is specified in a single instruction word.

Referring to Figure 1, the Auxiliary register (address 00) is the implied source of the second argument for ADD, AND or XOR operations.

IVL and IVR are write-only registers used only as a destination. They have addresses and are treated as registers, but in reality they do not exist. When IVL is specified as a destination or the D field = $20\text{-}27_8$, then LB = 'low', RB = 'high' are generated; when IVR is specified as a destination or the D field = $30\text{-}37_8$, then RB = low, LB = 'high' are generated.

When IVL or IVR is specified as the destination in an instruction, SC is also activated and data is placed on the IV bus. If IVL or IVR is specified as a source of data, the source data is all zeroes.

TYPICAL SYSTEM CONFIGURATION

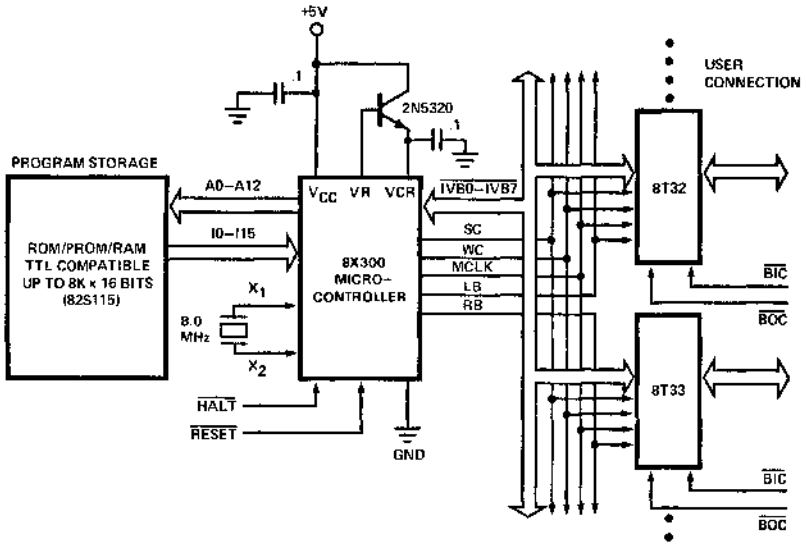


Figure 2

INSTRUCTION AND IV BUS DATA INPUT	DATA PROCESSING	ADDRESS AND IV BUS CHANGING	ADDRESS AND IV BUS DATA VALID MCLK=HIGH
← ¼ cycle →	← ¼ cycle →	← ¼ cycle →	← ¼ cycle →

Table 2 INSTRUCTION CYCLE

INSTRUCTION SEQUENCE CONTROL

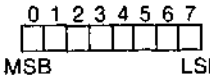
The Address Register and Program Counter are used to generate addresses for accessing an instruction. The Address Register is used to form the instruction address, and in all but 3 instructions (XEC, NZT, and JMP) the address is copied into the Program Counter. The instruction address is formed in 1 of 3 ways:

1. For all instructions but the JMP, XEC, and a satisfied NZT, the Program Counter is incremented by 1 and placed in the Address Register.
2. For the JMP instruction, the full 13-bit address field from the JMP instruction is placed into the Address Register and copied into the Program Counter.
3. For the XEC and NZT instructions, the high order 5- or 8-bits of the Program Counter are combined with 8- or 5-lower-order bits of ALU output (XEC or NZT) and placed in the Address Register. For the NZT instruction, it is also copied into the Program Counter.

INSTRUCTION SET

The 8X300 Microcontroller has a repertoire of 8 instruction classes which allow the user to test input status lines, set or reset output control lines, and perform high speed input/output data transfers. All instructions are 16 bits in length and each is fetched, decoded and executed in 250ns.

Data is respresented as an 8-bit byte; bit positions are numbered from left to right, with the least significant bit in position 7.



Within the 8X300, all operations are performed on 8-bit bytes. Arithmetic operations use 8-bit, unsigned 2's complement arithmetic.

INSTRUCTION FORMATS

The general 8X300 instruction format is:

0 1 2	3 4 5 6 7 8 9 10 11 12 13 14 15
Op Code	Operand(s)

All instructions are specified by a 3-bit Operation (Op Code) field. The operand may consist of the following fields: Source (S) field, Destination (D) field, Rotate/Length (R/L) field, Immediate (I) field, or (Program Storage) Address (A) field.

The instructions are divided into 5 format types, based on the Op Code and the Operand(s), as shown in Figure 3.

OPERATION	FORMAT	RESULT	NOTES
MOVE	I,II	Content of data field specified by {S, R/L} replaces data in field specified by {D, R/L.}	If S and D both are registers, then R/L specifies a right rotate of the register specified by S.
ADD	I,II	Sum of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L}.	
AND	I,II	Logical AND of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L.}	
XOR	I,II	Logical exclusive OR of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L.}	
XMIT	III,IV	The literal value I replaces the data in the field specified by {S,L.}	If S is an I/O address then I is limited to range 00-37. Otherwise I is limited to range 000-377.
NZT	III,IV	If the data in the field specified by {S, L} equals zero, perform the next instruction in sequence. If the data specified by {S,L} is not equal to zero, execute the instruction at address determined by using the literal I as an offset to the Program Counter.	If S is an I/O address then I is limited to range 00-37. Otherwise I is limited to range 000-377.
XEC	III,IV	Perform the instruction at address determined by applying the sum of the literal I and the data specified by {S, L} as an offset to the Program Counter. If that instruction does not transfer control, the program sequence will continue from the XEC instruction location.	The offset operation is performed by reducing the value of PC to the nearest multiple of 32 (if I = 00-37) or 256 (if I = 000-377) and adding the offset.
JMP	V	The address value A replaces contents of the Program Counter.	A limited to the range 0-17777$_8$.

Table 3 8X300 INSTRUCTION SUMMARY

INSTRUCTION FORMATS

OPERATIONS
(REGISTER TO REGISTER)
MOVE AND
ADD XOR

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
OP CODE	S	R (ROTATE)	D

Type I

OPERATIONS
(REGISTER TO I/O, I/O TO REGISTER, I/O TO I/O)
MOVE ADD
AND XOR

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
OP CODE	S	L (LENGTH)	D

Type II

OPERATIONS

XEC XMIT* NZT

0 1 2	3 4 5 6 7	8 9 10 11 12 13 14 15
OP CODE	S	I

Type III

OPERATIONS

XEC XMIT* NZT

0 1 2	3 4 5 6 7	8 9 10	11 12 13 14 15
OP CODE	S	L (LENGTH)	I

Type IV

OPERATIONS

JMP

0 1 2	3 4 5 6 7 8 9 10 11 12 13 14 15
OP CODE	A

Type V

*NOTE
If XMIT, S actually represents the destination

Figure 3

INSTRUCTION FIELDS

Op Code Field (3-Bit Field)

The Op Code field is used to specify 1 of 8 8X300 instructions as shown in Table 4.

OP CODE OCTAL VALUE	INSTRUCTION	
0	MOVE	S,R/L,D
1	ADD	S,R/L,D
2	AND	S,R/L,D
3	XOR	S,R/L,D
4	XEC	I,R/L,S or I,S
5	NZT	I,R/L,S or I,S
6	XMIT	I,R/L,D or I,D
7	JMP	A

Table 4 OP CODE FIELD OCTAL ASSIGNMENTS

S,D Fields (5-Bit Fields)

The S and D fields specify the source and destination of data for the operation defined by the Op Code field. The Auxiliary Register is an implied second source for the instructions ADD, AND and XOR, each of which require two source fields. That is, instructions of the form,

ADD X, Y

imply a third operand, say Z, located in the Auxiliary Register so that the operation which takes place is actually X + Z, with the result stored in Y.

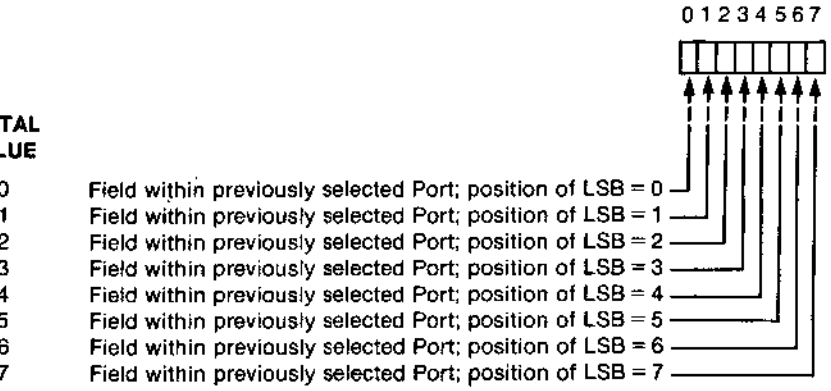
The S and/or D fields may specify a register, or a 1 to 8-bit I/O field. S and D field value assignments in octal are shown in Table 5.

0_8-17_8 is used to specify 1 of 7 working registers (R1-R6, R11), the Auxiliary Register, the Overflow Register, or IVL and IVR write-only registers.

OCTAL VALUE		OCTAL VALUE	
00	AUX-Auxiliary Register	10	OVF-Overflow register-Used only as a source
01	R1	11	R11
02	R2	12	Unassigned
03	R3	13	Unassigned
04	R4	14	Unassigned
05	R5	15	Unassigned
06	R6	16	Unassigned
07	IVL Register-Left Bank I/O address register. Used only as a destination.	17	IVR Register-Right Bank I/O address register. Used only as a destination.

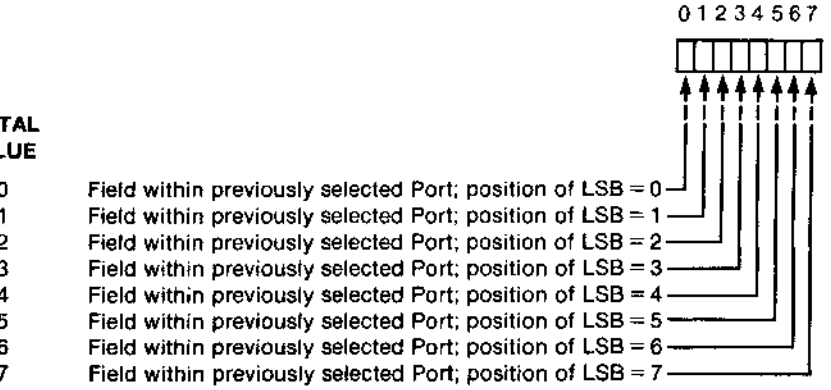
a. Register Specification

20_8-27_8 is used to specify the least significant bit of a variable length field within the I/O Port previously selected by the IVL register. The length of the field is determined by R/L.



b. Left Bank I/O Field Specification

30_8-37_8 is used to specify the least significant bit of a variable length field within the I/O Port previously selected by the IVR Register. The length of the field is determined by R/L.



c. Right Bank I/O Field Specification

Table 5 S AND D FIELD OCTAL ASSIGNMENTS

R/L Field (3-Bit Field)

The R/L field performs one of two functions, specifying either a field length (L) or a right rotation (R). The function it specifies for a given instruction depends upon the contents of the S and D fields:

A. When both S and D specify registers, the R/L field is used to specify a right rotation of the data specified by the S field. (Rotation occurs on the bus and not in the source register.) The register source data is right rotated within one instruction cycle time independent of the number of bit positions specified in the R/L field.
B. When either or both the S and D fields specify a variable length I/O data field, the R/L field is used to specify the length of that data field.
C. R/L field assignments are shown in Table 6.

R/L FIELD OCTAL VALUE	SPECIFICATION
0	Field Length = 8 Bits
1	Field Length = 1 Bit
2	Field Length = 2 Bits
3	Field Length = 3 Bits
4	Field Length = 4 Bits
5	Field Length = 5 Bits
6	Field Length = 6 Bits
7	Field Length = 7 Bits

Table 6 R/L FIELD OCTAL ASSIGNMENTS

I Field (5/8-Bit Field)

The I field is used to load a literal value (contained in the instruction) into a register, or a variable I/O data field, or to modify the low order bits of the Program Counter.

The length of the I field is based on the S field in XEC, NZT, and XMIT instructions, as follows:

A. When S specifies a register, the literal I is an 8-bit field (Type III format).
B. When S specifies variable I/O data field, the literal I is a 5-bit field (Type IV format).

A Field (13-Bit Field)

The A field is a 13-bit Program Storage address field. This allows the 8X300 to directly address 8192 instructions.

REGISTER OPERATIONS

When a register is specified as the source and a variable I/O data field is specified as the destination, the low order bits of the results of the instructions MOVE, ADD, XOR are merged with the original destination data.

When an I/O data field of 1 to 8 bits is specified as the source, and a register as the destination, the 8-bit result of the operations MOVE, ADD, AND, XOR is stored in the register. The operations ADD, AND, XOR actually use the I/O data field (1 to 8 bits) with leading zeros to obtain 8-bit source data for use with the 8-bit AUX data during the operation.

IVL and IVR are write-only pseudo registers, and therefore can be specified as destination fields only. Operations involving IVL and IVR as sources are not possible. For example, it is not possible to increment IVR or IVL in a single instruction, and the contents of IVL or IVR cannot be transferred to a working register, or I/O Port.

The OVF (Overflow) Register can only be used as a source field; it is set or reset *only* by the ADD instruction.

ADDRESSING DATA ON THE INTERFACE VECTOR

I/O data fields are implemented via general purpose 8-bit I/O registers called Interface Vector (IV) Bytes. The IV registers serve to select IV bytes. In order for an instruction to access (read or write) an I/O data field, the address must be output to the IVL or IVR registers.

Thus, two instructions are required to operate on an Interface Vector byte.

```
XMIT    ADDRESS, IVL
MOVE    LB, RB
```

Each of the two IV registers (IVL and IVR) may be set to select an IV byte, therefore two I/O ports may be active at one time—one on the Right Bank (IVR) and one on the Left Bank (IVL). Data may be input and output in one instruction following the selection of IV bytes:

```
XMIT    ADDRESS1, IVL
XMIT    ADDRESS2, IVR
ADD     LB, RB
```

Once the IV byte is selected (addressed) it will remain selected until another address is output to the same IV register. Since an IV register (IVL, IVR) can be used only as a destination field of an instruction, any instruction sending data to IVL or IVR can be used to select an IV byte.

From the user's standpoint, however, all IV byte outputs can be read by an external device regardless of whether they are selected or not.

The address range of IVL and IVR is $0\text{-}255_{10}$.

INSTRUCTION DESCRIPTIONS

The following instruction descriptions employ MCCAP (the 8X300 Cross Assembly Program) programming notation. This notation varies somewhat from the instruction descriptions provided in Tables 3 through 5. Thus, for example, explicit L field definition, as shown in Table 3 and Table 4 is not required by MCCAP instructions; MCCAP can create appropriate variable field addresses from information contained in Data Declaration statements which may be provided by the programmed at the beginning of his program.

The 8X300 instruction set is described below with examples shown in Figures 4 through 11.

MOVE S,D or MOVE S(R),D

Format: Type I, Type II

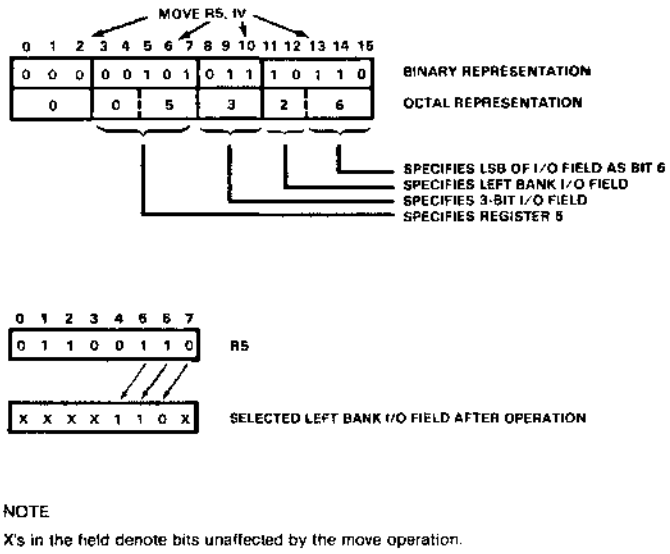
Operation: (S)→(D)

Description

Move data. The contents of S are transferred to D; the contents of S are unaffected. If both S and D are registers, R/L specifies a right rotate of the source data before the move. Otherwise, R/L specifies the length of the source and/or destination I/O data field. If the MOVE is between Left Bank and Right Bank I/O field, an 8-bit field must always be moved.

Example

Store the least significant 3 bits of register 5 (R5) in bits 4, 5 and 6 of the I/O Port previously addressed by the IVL register. See Figure 4.



NOTE

X's in the field denote bits unaffected by the move operation.

Figure 4

ADD S,D or ADD S(R),D

Format: Type I, Type II

Operation: (S) + (AUX)→D
Carry→OVF

Description

Unsigned 2's complement 8-bit addition. The contents of S are added to the contents of the Auxiliary Register. The result is stored in D; OVF is set to the value of the carry. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the operation. Otherwise, R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as the destination.

Example

Add the contents of R1 rotated 4 places to AUX and store the result in R3. See Figure 5.

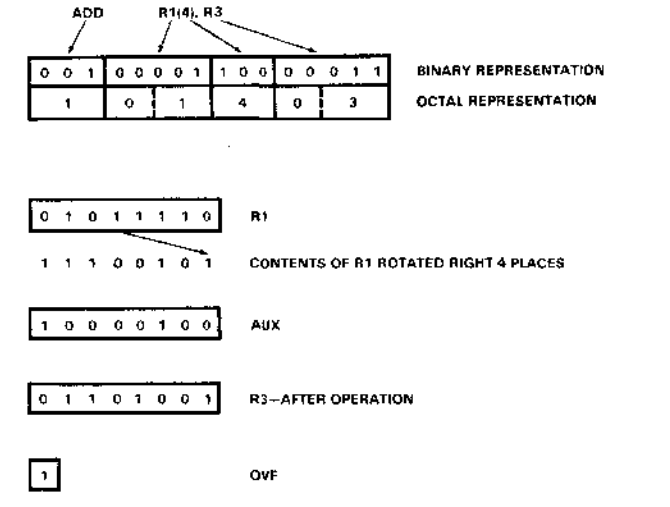


Figure 5

AND S,D or AND S(R),D

Format: Type I, Type II

Operation: (S) ∧ (AUX)➔D

Description

Logical AND. The AND of the source field and the Auxiliary Register is stored into the destination. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the AND operation. Otherwise R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as a destination.

Example

Store the AND of the selected right bank I/O field and AUX in R4. The right bank data field is called WSBCD and is 4 bits long and located in bits 2, 3, 4 and 5. See Figure 6.

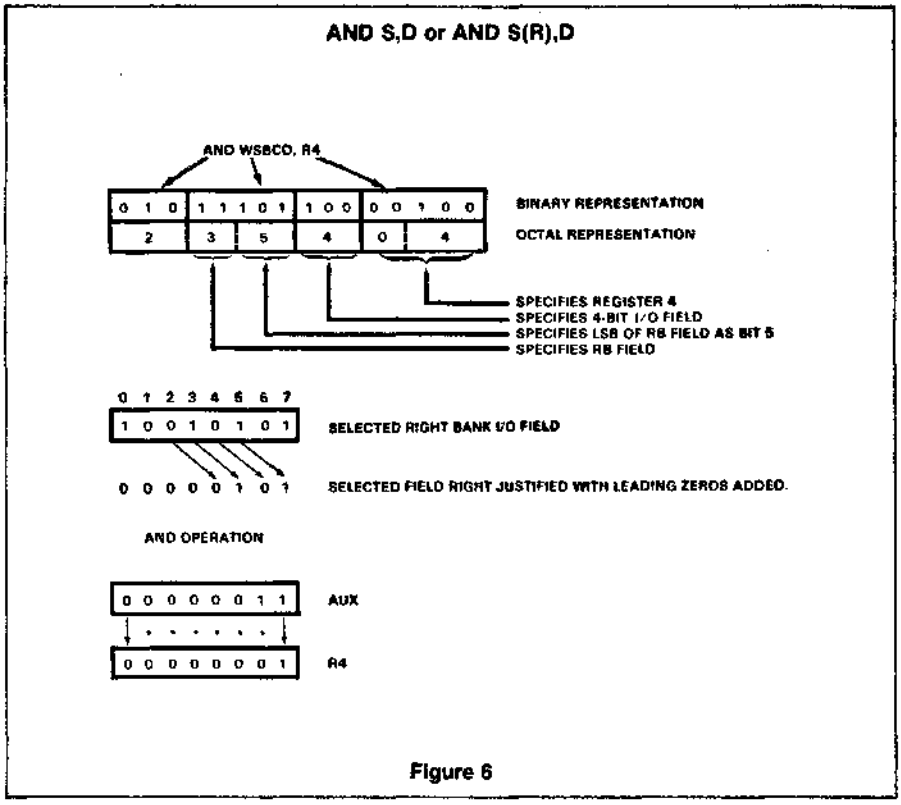


Figure 6

XOR S,D or XOR S(R),D

Format: Type I, Type II

Operation: (S) ⊕ (AUX)➔D

Description

Exclusive-OR. The Exclusive-OR of the source field and the Auxiliary Register is stored in the destination. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the XOR operation. Otherwise R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as a destination.

Example

Replace the selected I/O data field with the XOR of the field and AUX. The I/O data field is called STATUS and is 5 bits in length and located in bits 3, 4, 5, 6 and 7 of left bank. See Figure 7.

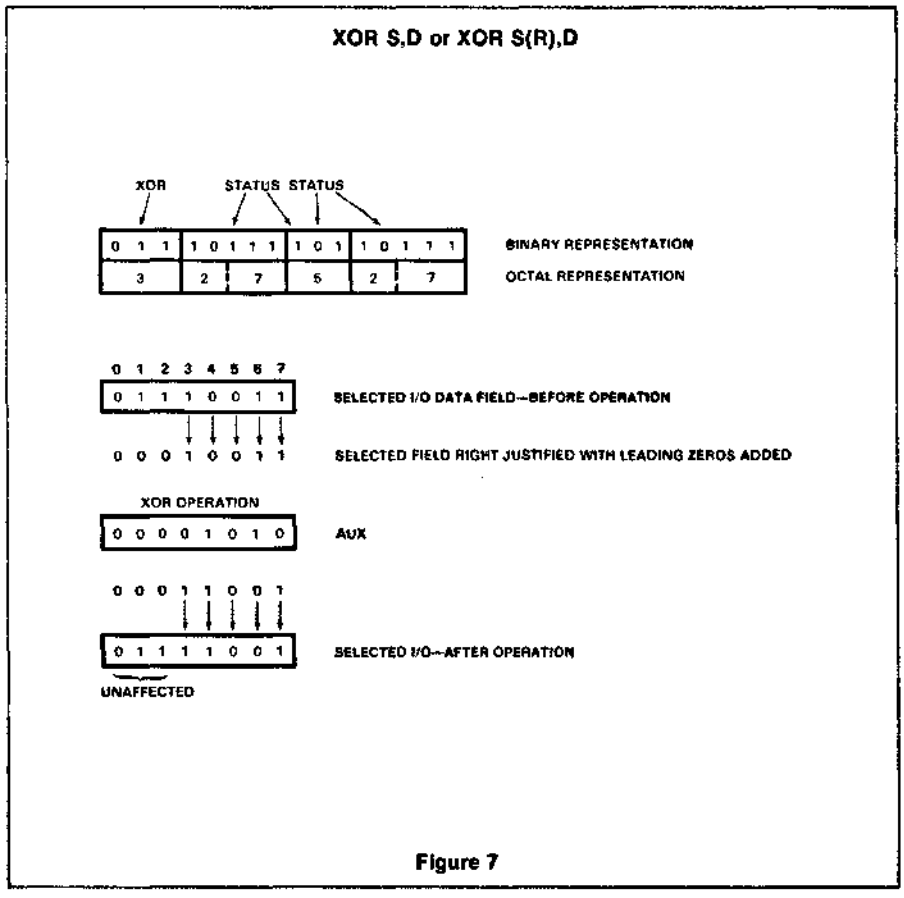


Figure 7

XEC I(S)

Format: Type III, Type IV

Operation

Execute instruction at the address specified by the Address Register with lower 5 or 8 bits replaced by (S) + I.

Description

Execute the instruction at the address determined by replacing the low order bits of the Address Register (AR) with the low order bits of the sum of the literal I and the contents of the source field. If S is a register, the low order 8 bits of AR are replaced; if S is an I/O data field, the low order 5 bits of AR are replaced, resulting in an execute range of 256 and 32 respectively. The Program Counter is not affected unless the instruction executed is a JMP or NZT (whose branch is taken).

Example

Execute one of n JMPs in a table of JMP instructions determined by the value of the selected I/O data field on the left bank. The table follows immediately after the XEC instruction and the I/O field is called INTERPT and is a 3-bit field located in bits 4, 5 and 6. See Figure 8.

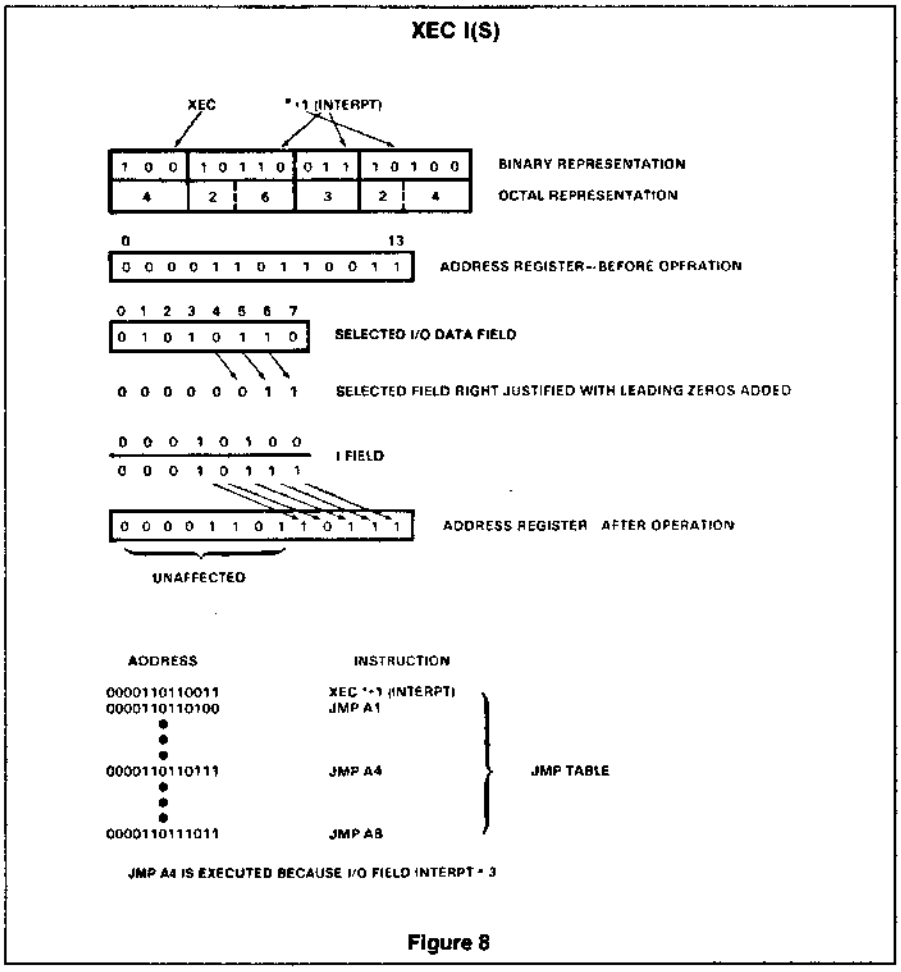


Figure 8

XMIT I,D

Format: Type III, Type IV

Operation: I→(D)

Description

Transmit literal. The literal field I is stored in D. If D is a register, an 8-bit field is transferred; if D is an I/O data field, up to a 5-bit field is transferred.

Example

Store the bit pattern 110 in the selected I/O data field on the right bank. The field name is VALUE and is located in bits 3, 4 and 5. See Figure 9.

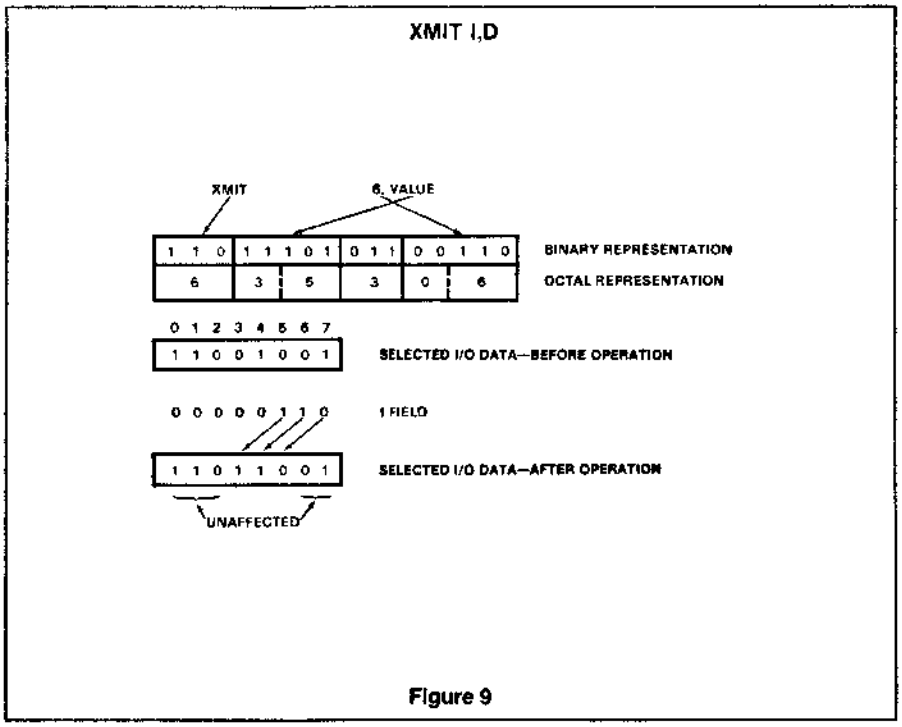


Figure 9

NZT S,I

Format: Type III, Type IV

Operation:

Non-Zero Transfer. If (S) ≠ 0, PC offset by I → PC; otherwise PC + 1 → PC.

Description

If the data specified by the S field is non-zero, replace the low order bits of the Program Counter with I. Otherwise, processing continues with the next instruction in sequence. If S is a register, the low order 8 bits of the PC are replaced; if S is an I/O data field, the low order 5 bits of the PC are replaced, resulting in an NZT range of 256 and 32 respectively.

Example

Jump to Program Address ALPHA if the selected left bank I/O field is non-zero. The field name is OVERFLO and it is a 1-bit field located in bit 3. See Figure 10.

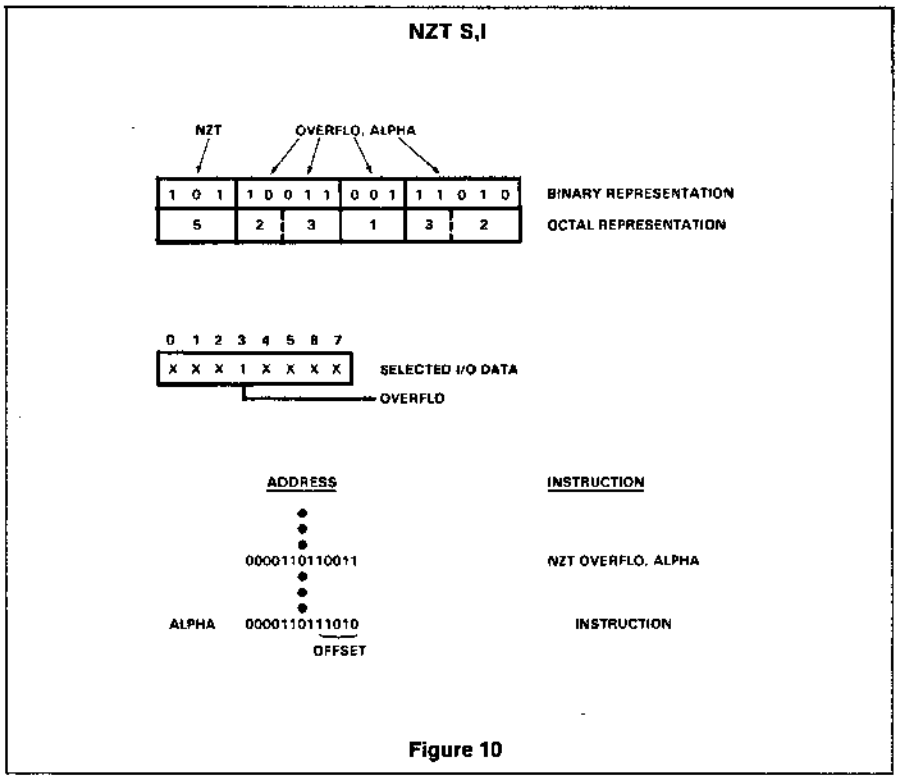


Figure 10

JMP A

Format: Type V

Operation: A → PC

Description

The literal value A is placed in the Program Counter and Address Register, and processing continues at location A. A has a range of 0-17777$_8$ (0-8191).

Example

Jump to location ALPHA (0000101110001). See Figure 11.

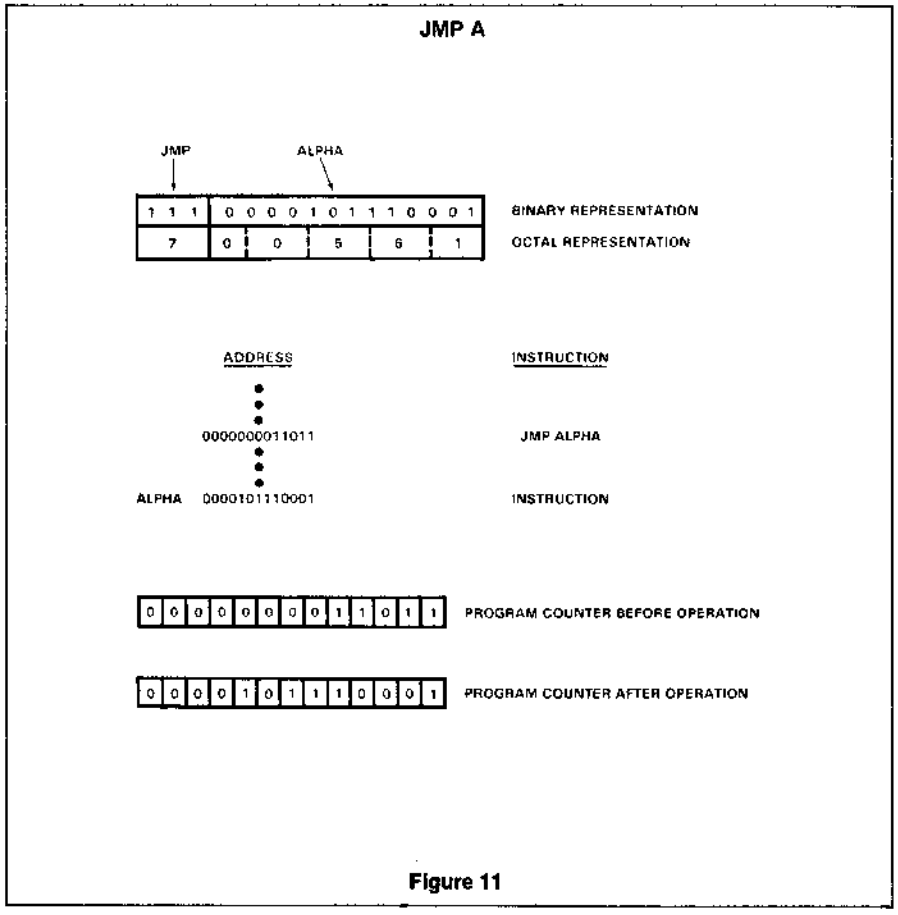


Figure 11

SYSTEM DESIGN USING THE 8X300 MICROCONTROLLER

Designing hardware around the 8X300 Interpreter reduces to selecting a program storage devicer (ROM, PROM, etc.), selecting I/O devices (IV byte, multiplexers, RAM, etc.), selecting clock mode (system driven or crystal controlled) and interfacing the Microcontroller to these components.

A specific example of a control system using the 8X300 Microcontroller is shown in Figure 12. Only 8 components—four 8T32 I/O Ports, one 8X350 RAM, two 82S215 ROMs, and an 8X300 are required to build this system which contains 512 words of program storage, 32 TTL I/O connection points, 256 bytes of working storage, and operates at a 250ns instruction cycle time.

Halt, Reset Signals

$\overline{\text{HALT}}$:
A low level at the $\overline{\text{HALT}}$ input stops internal operation of the Microcontroller at the start of the next instruction after $\overline{\text{HALT}}$ is applied (quarter cycle after MCLK). Since $\overline{\text{HALT}}$ is sampled at the start of each instruction cycle it is possible to prevent a cycle by applying $\overline{\text{HALT}}$ early in that cycle. $\overline{\text{HALT}}$ does not inhibit MCLK or affect any internal registers. Normal operation begins with the next complete cycle after the $\overline{\text{HALT}}$ input goes high.

$\overline{\text{RESET}}$:
A low level at the $\overline{\text{RESET}}$ input sets the program counter and address register to zero. While $\overline{\text{RESET}}$ is low MCLK is inhibited. If $\overline{\text{RESET}}$ is applied during the last 2 quarter cycles, the MCLK during that cycle may be shortened. $\overline{\text{RESET}}$ should be applied for 1 full instruction cycle time to assure proper operation. When $\overline{\text{RESET}}$ input goes high an MCLK occurs prior to the resumption of normal processing. $\overline{\text{RESET}}$ does not affect the other internal registers.

SYSTEM TIMING

In systems with fast instruction cycle times, most Microcontroller delays are strictly determined by internal gate propagation delays.

EXAMPLE OF CONTROL SYSTEM

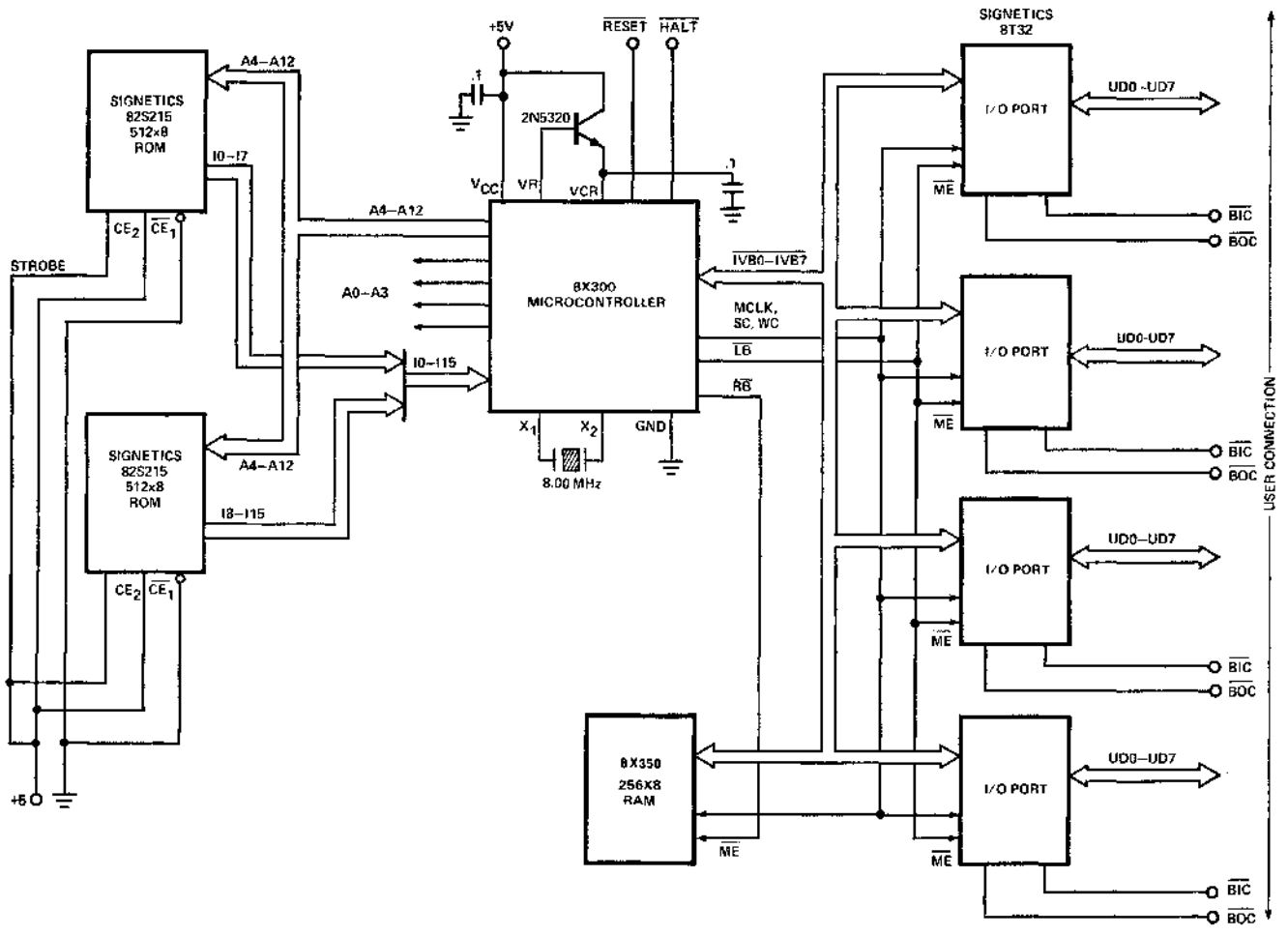


Figure 12

Since some events are constrained to occur in certain quarter cycles, as system cycle times become slower, the delays will appear to increase due to gating with internal clocks. In the table of AC Electrical Characteristics, 2 columns are used: 1 to denote times which occur due to internal clock intervention and 1 to denote minimum delays for fast cycle times.

When using Signetics 8T32 I/O Ports, selection of instruction cycle time involves calculating the maximum program storage access time. Assuming the instruction is available when MCLK falls, the I/O control lines are stable 35ns later. Signetics 8T32's require another 35ns to disable a previously selected port and enable the desired port (assumes a change in bank signals). A 10ns margin has been added to the 8T32 enable for this evaluation to reflect the fact that most systems will have more capacitive loading than the 50pF test condition in the 8T32 specification and to allow for line delays.

The system instruction cycle time for normal systems such as shown in Figure 12 is determined by Microcontroller propagation delays, program storage access time, and port output enable times. Instruction cycle time is normally constrained by one or more of the following conditions:

1. Instruction to LB/RB (input phase) + I/O Port output enable: $T_{OE} \leq$ ½cycle - 55ns (Figure 13).
2. Program storage access time + instruction to LB/RB (input phase) + I/O Port output enable and IV data (input phase) to address ≤ instruction cycle time (Figure 14).
3. Program storage access time + instruction to address ≤ instruction cycle time (Figure 15).

The first constraint can be used to determine the minimum cycle time. Using the inequality 35ns + 35ns ≤ ½ cycle - 55ns implies ½ cycle ≥ 125ns or an instruction time of 250ns.

Program storage access time for a 250ns instruction cycle can be calculated from the second constraint. Noting that the specification for IV data (input phase) to address is 115ns: Program storage access time + 35ns + 35ns + 115ns ≤ 250ns implies program storage access time ≤ 65ns.

The third constraint can be used to verify the maximum program storage access time. Noting that the specification for instruction to address is 185ns: Program storage access time + 185ns ≤ 250ns confirms that program storage access time 65ns is satisfactory.

SYSTEM INSTRUCTION CYCLE TIME – INSTRUCTION TO IV DATA TIMES

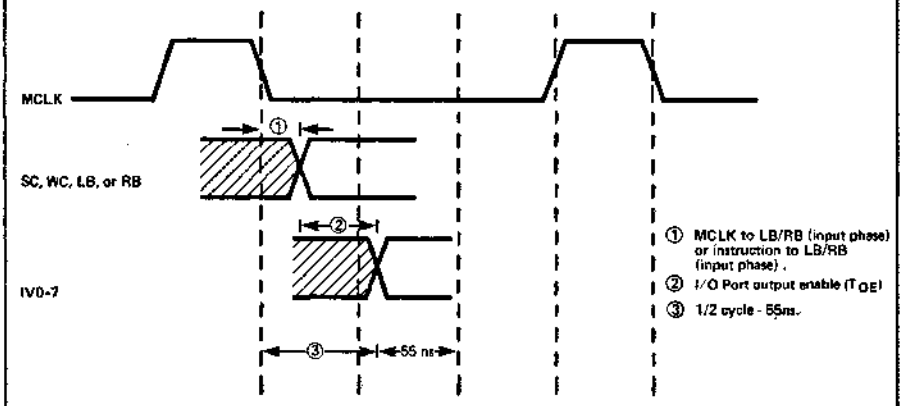


Figure 13

INSTRUCTION CYCLE TIME

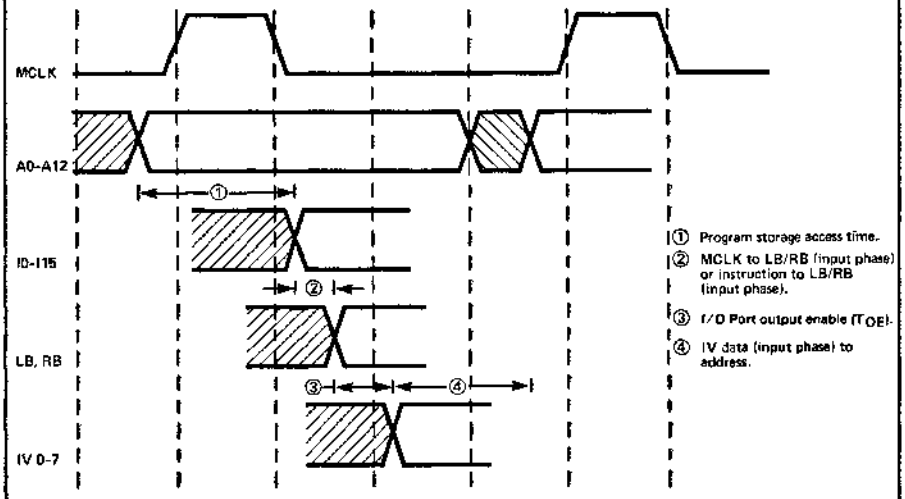


Figure 14

PROGRAM STORAGE ACCESS TIME

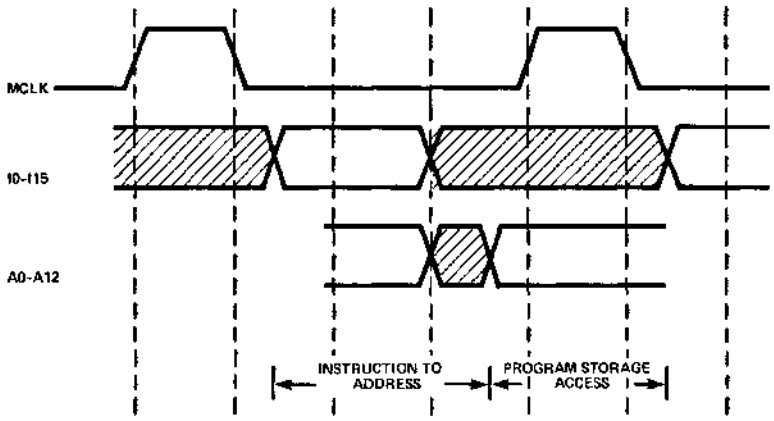


Figure 15

System Clock

The Micrcontroller has an integrated oscillator which generates all necessary clock signals. The oscillator is designed to connect directly to a series resonant quartz crystal via pins X1 and X2. The crystal resonant frequency, f, is related to the desired cycle time, T, by the relationship f = 2/T. For a 250ns system, f = 8.00MHz.

Type:	Fundamental mode, series resonant
Impedance at Fundamental:	35 ohms maximum
Impedance at harmonics and spurs:	50 ohms minimum

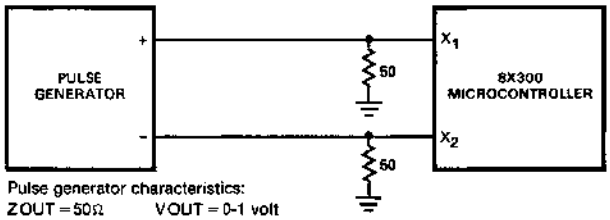
Table 7 CRYSTAL CHARACTERISTICS

In lower speed applications where the cycle time need not be precisely controlled, a capacitor may be connected between X1 and X2 to drive the oscillator. Approximate capacitor values are given in Table 8. If cycle time is to be varied, X1 and X2 should be driven from complementary outputs of a pulse generator. Figure 16 shows a typical configuration. For systems where the Interpreter is to be driven from a master clock the X1 and X2 lines may be interfaced to TTL logic as shown in Figure 17.

Cx,pF	CYCLE TIME
100	300ns
200	500ns
500	1.1μs
1000	2.0μs

Table 8 CLOCK CAPACITOR VALVES

CLOCKING WITH A PULSE GENERATOR

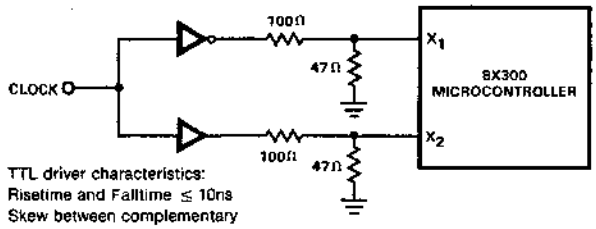


Pulse generator characteristics:
ZOUT = 50Ω VOUT = 0-1 volt
Risetime ≤ 10ns ≤ Skew ≤ 10ns
Complementary outputs

NOTE: All resistors values are typical and in ohms.

Figure 16

CLOCKING WITH TTL



TTL driver characteristics:
Risetime and Falltime ≤ 10ns
Skew between complementary outputs ≤ 10ns

Figure 17

AC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ± 5% and 0°C ≤ T_A < 70°C

DELAY DESCRIPTION	PROPAGATION DELAY TIME	CYCLE TIME LIMIT
X1 falling edge to MCLK (driven from external pulse generator)	75ns	
MCLK to SC/WC falling edge (input phase)	25ns	
MCLK to SC/WC rising edge (output phase)		½ cycle + 25ns
MCLK to $\overline{LB}/\overline{RB}$ (input phase)	35ns	
Instruction to $\overline{LB}/\overline{RB}$ output (input phase)	35ns	
MCLK to $\overline{LB}/\overline{RB}$ (output phase)		½ cycle + 35ns
MCLK to IV data (output phase)	185ns	½ cycle + 60ns
IV data (input phase) to IV data (output phase)	115ns	
Instruction to Address	185ns	½ cycle + 40ns
MCLK to Address	185ns	½ cycle + 40ns
IV data (input phase) to Address	115ns	
MCLK to IV data (input phase)		½ cycle - 55ns
MCLK to Halt falling edge to prevent current cycle		¼ cycle - 40ns
Reset rising edge to first MCLK		0 to 1 cycle

NOTE
1. Reference to MCLK is to the falling edge when loaded with 300pF.
2. Loading on Address lines is 150pF.

TYPICAL INSTRUCTION CYCLE TIMING

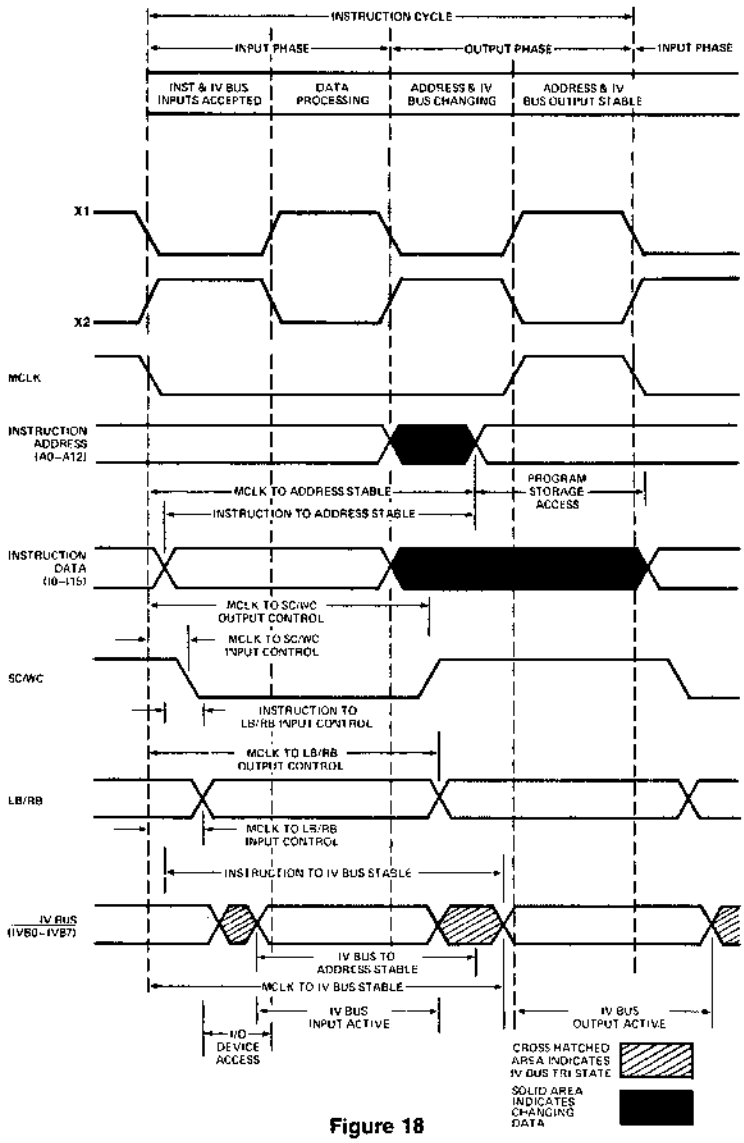


Figure 18

DC ELECTRICAL CHARACTERISTICS

Limits apply for $V_{CC} = 5V \pm 5\%$ and $0°C < T_A < 70°C$ unless specified otherwise.

PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
V_{IH}	High level input voltage					
	X1,X2		.6			V
	All others		2			V
V_{IL}	Low level input voltage					
	X1,X2				.4	V
	All others				.8	V
V_{IC}	Input clamp voltage (Note 1)	$V_{CC} = 4.75V$, $I_I = -10mA$			-1.5	V
I_{IH}	High level input current					
	X1,X2	$V_{CC} = 5.25V$, $V_{IH} = .6V$		2700		μA
	All others	$V_{CC} = 5.25V$, $V_{IH} = 4.5V$		<1	50	μA
I_{IL}	Low level input current					
	X1,X2	$V_{CC} = 5.25V$, $V_{IL} = .4V$		-2500		μA
	$\overline{IVBO}$-7			-140	-200	μA
	IO-I15	$V_{CC} = 5.25V$, $V_{IL} = .4V$		-880	-1600	μA
	$\overline{HALT}$, $\overline{RESET}$	$V_{CC} = 5.25V$, $V_{IL} = .4V$		-230	-400	μA
V_{OL}	Low level output voltage					
	A0-A12	$V_{CC} = 4.75V$, $I_{OL} = 4.25mA$		.35	.55	V
	All others	$V_{CC} = 4.75V$, $I_{OL} = 16mA$		.35	.55	V
V_{OH}	High level output voltage	$V_{CC} = 4.75V$, $I_{OH} = 3mA$	2.4			V
I_{OS}	Short circuit output current (Note 2)	$V_{CC} = 5.25V$	-30		-140	mA
V_{CC}	Supply voltage		4.75	5	5.25	V
I_{CC}	Supply current	$V_{CC} = 5.25V$			1.60	mA
I_{REG}	Regulator control	$V_{CC} = 5.0V$	-14		-21	mA
I_{CR}	Regulator current (Note 3)				290	mA
V_{CR}	Regulator voltage (Note 3)		2.2		3.2	V

NOTES

1. Crystal inputs X1 and X2 do not have clamp diodes.
2. Only one output may be grounded at a time.
3. $V_{CC} = 5.25V$, HALT = RESET = ADDRESS = IVX = 0.0V, all other pins open.

ABSOLUTE MAXIMUM RATINGS

PARAMETER	RATING	UNIT
V_{CC} Supply voltage	7	V
X1,X2 Crystal input voltage	2	V
Others Logic input voltage	5.5	V

SUPPORT PRODUCTS

DESIGNER'S EVALUATION KIT FOR FIXED INSTRUCTION BIPOLAR MICROPROCESSOR — 8X300KT100SK

DESCRIPTION

The Signetics 8X300 Fixed Instruction Bipolar Microprocessor provides new levels of high performance to microprocessor applications not previously possible with MOS technology.

In the majority of cases, the choice of a bipolar microprocessor slice, as opposed to an MOS device, is based on speed. The 8X300 processor, combined with high-speed memory and I/O devices, is capable of executing all instruction in 250ns.

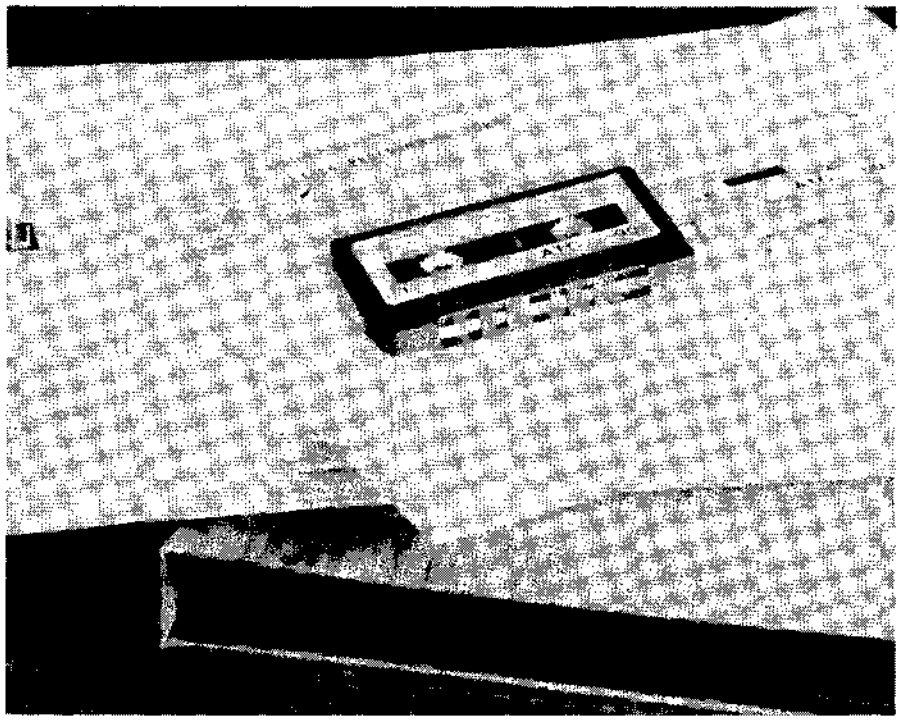
The 8X300 is optimized for control and data movement applications. It has a 13-bit address bus for selecting instructions from program storage and a separate input bus for entering a 16-bit instruction words. Data handling and I/O device addressing are accomplished via the 8-bit Interface Vector (IV) bus. The IV bus is supported by four additional control lines and a clock.

The unique features of the 8X300 IV bus and instruction set permit 8-bit parallel data to be rotated or masked before undergoing arithmetic or logical operations. Then, the data may be shifted and merged into any set of from 1 to 8 contiguous bits at the destination. The entire process of input, shifting, processing and output is done in 1 instruction cycle time. The 250ns cycle time makes the 8X300 ideally suited for high-speed applications.

The evaluation board contains all the elements which a designer needs to judge the suitability of the 8X300 for his systems applications. Included with the 8X300 are 4 I/O ports for external device interface, 256 bytes of temporary (working) data storage, and 512 words of program storage, all properly connected to the 8X300 to allow immediate exercising of the board. For this purpose, the PROMs are preprogramed with the I/O control, RAM control, and RAM integrity diagnostic programs. With the remaining PROM space, the designer may enter his own benchmark, test, or development routines.

The board design allows complete flexibility in access to the address, instruction, and IV busses as well as all controls and signals of the 8X300. The IV bus, I/O port user connection, clock signals control lines, address bus and instruction bus are wired to output pins, the board edge connector and flat cable connectors.

The board layout permits variations and/or expansions of the basic design. In addition to the access to all signals for transfer off the board, a wire wrap area is provided so that the designer may add to the board circuitry as he desires. The addition may include memory, additional interfaces, or special circuits which meet specific user requirements.



Controls are also provided for diagnostic and instructional purposes by allowing various operating modes. In the WAIT mode, the program may be single stepped for ease of checkout. The one-shot instruction jamming allows control of the program start location, changes of program flow, changing or examining the internal registers, or testing of simple sequences. The repeated instruction jamming provides a means of repetitive execution of an instruction so that the I/O bus and the control lines may be examined without software changes. In both of these jam cases, the jammed instruction is selected by board-mounted switches.

FEATURES

- **250ns CPU with Crystal**
- **4 I/O Ports (32 Lines)**
- **256 Bytes Data Storage**
- **512 Words Program Storage**
- **Run/Wait Control**
- **Single Step**
- **Instruction Jamming**
  One Shot Instruction Jam
  Repeated Jam
- **All Buses to Output Pins**
- **Firmware Diagnostics**
- **Wire-Wrap Area**
- **Edge Connector**
- **Flat Cable Connectors**
- **Wire Wrap Posts for Bus Lines**

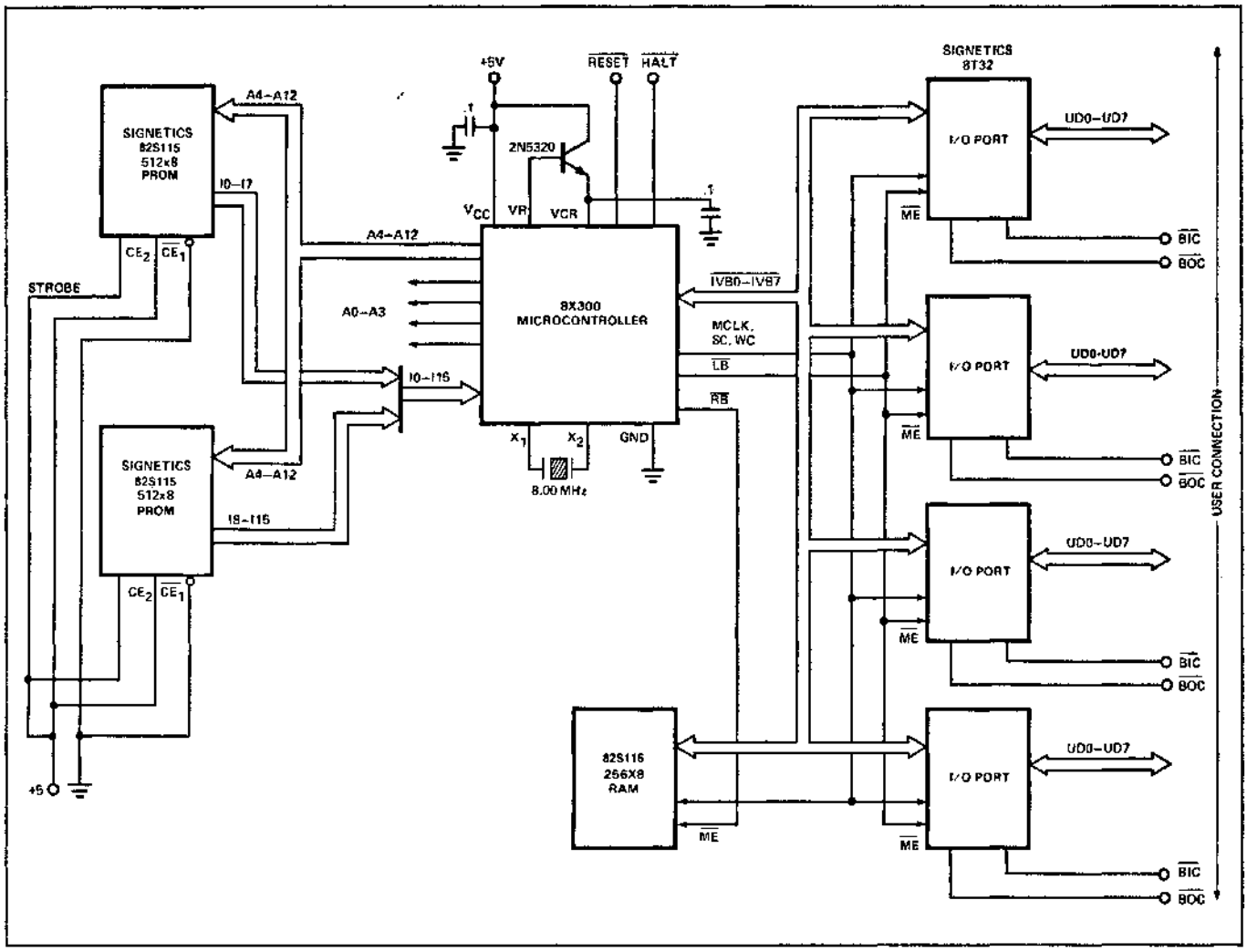
CONTENTS

1 ea— 8X300
8 ea— 82S116 (256 x 1 RAM)
2 ea— 82S115 (512 x 8 PROM)
4 ea— 8T32 Addressable Bidirectional I/O Port
1 ea— 8T31 (Bidirectional I/O Port)
2 ea— 8T26A (Quad Bus Transceiver)
4 ea— 74157 (Quad 2-Input Data Selector)
2 ea— 7474 (Dual D Flip Flop)
2 ea— 7400 (Quad Nand Gate)
1 ea— 7427 (3-Input NOR Gate)
1 ea— P.C. Board
Misc. Parts
1 ea— Introductory Manual, assembly instructions, code listings and schematics

AVAILABILITY

Immediate delivery from Signetics Rep. or Distributors.

BLOCK DIAGRAM



Auxiliary Circuits

The 8X300 can be used with any bipolar (or TTL-compatible) ICs. It can directly address 8192 program instruction locations and up to 512 I/O ports. The memory paging feature may be employed for larger working storage. Typical auxiliary circuits include:

Program Storage		82S115 (512x8 PROM)
I/O	8T32/33	(8-Bit Synchronous Bidirectional I/O Port)
	8T35/36	(8-Bit Asynchronous Bidirectional I/O Port)
	8T31	(8-Bit Bidirectional I/O Port)
	8T39	(Quad Bus Extender)
Working Storage		82S116 RAM

DESCRIPTION

The 8X300 Programming Course is an audio/visual course designed to assist system logic designers with little or no previous microprocessor programming experience to confidently and effectively design and generate software for his own application.

The 8X300 Programming Course consists of a series of 13 modular lessons. Each lesson consists of an audio tape and interactive course materials. Two short lessons describe 8X300 hardware features, internal architecture, systems configuration and interfaces, support devices and introduce a variety of software and prototype development tools. The user is then introduced to software and program organization concepts and terminology followed by detailed specification and analysis of instruction formats and operations in six short modules. Opportunity is provided to those students with basic programming skills to bypass one or more of these lessons. A separate lesson on program management and recordkeeping requirements is a natural bridge between instruction details and multi-instruction sequences. The remaining four modules of the course describe a variety of applications processes and specific techniques for effective program design and management. Particular emphasis is given to practical approach and choice of optional solutions. Throughout the course, student learning is reinforced and measured through integrated examples and exercises.

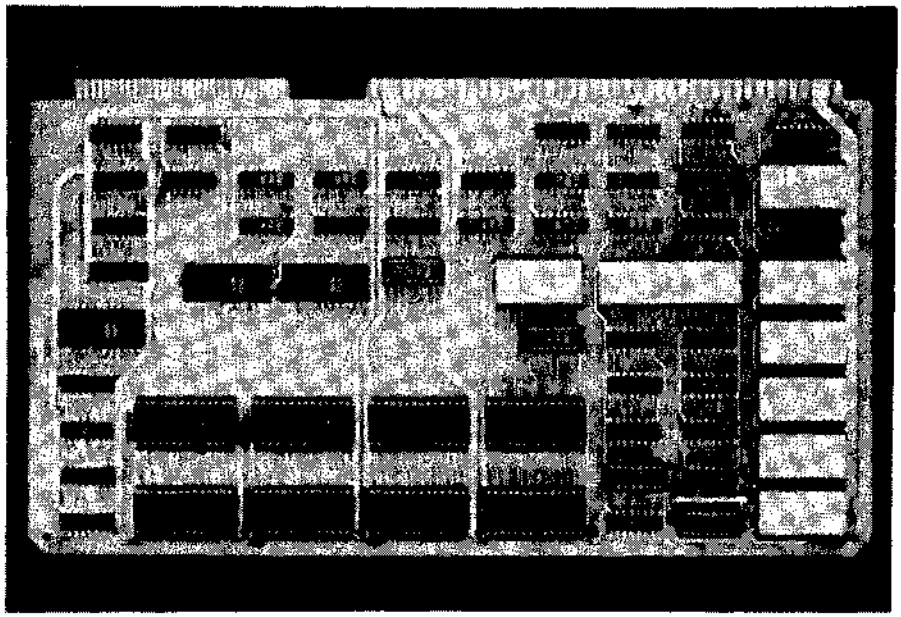
FEATURES

- **Audio/visual**
- **Self-paced course**

COURSE CONTENTS

10 Audio tapes in flat-pack album
6 Envelopes containing course notes
1 Folder containing data sheets and application notes
1 Tablet of 8X300 programming forms

During the course of training, the audio cassette tapes are used in conjunction with the course notes to provide the audio/visual aid. Exercises and pretests are also included at the end of each lesson to assist the student in understanding the course material and at the same time set a guideline for self-pacing.



CONTENTS DESCRIPTION

8X300 PROGRAMMING COURSE OVERVIEW

8X300 Hardware

Lesson 1 Features and architecture
Lesson 2 Support devices and prototype development tools

8X300 Software Basics

Lesson 3 Introduction to software
Lesson 4 Instruction formats
Lesson 5 Instruction operands

8X300 Instruction Commands

Lesson 6 Data movement
Lesson 7 Arithmetic logic
Lesson 8 Program control
Lesson 9 Record-keeping

Multi-Instruction Applications

Lesson 10 Part 1
Lesson 11 Part 2

Subroutine Calls And Returns

Lesson 12 Part 1
Lesson 13 Part 2

COURSE ABSTRACT

Lesson 1—8X300 Features and architecture
A brief description of 8X300 features and capabilities is followed by details of architecture and data flow through the 8X300. All pinouts are defined, and timing considerations relative to a minimum 8X300 system configuration are then illustrated and explained.

1. Design objectives.
2. 8X300 applications and usage.
3. Capabilities and limitations.
4. Power requirement and support circuit.
5. External timing requirements.
6. Data flow and throughput; processing time.
7. 8X300 architecture.
8. Instruction cycle phases and timing.
9. Details of the Interface Vector and I/O control.
10. Internal register definition and usage.
11. 8X300 minimum system configuration.

Audio Tape: Approximately 33 minutes
Student Completion: Approximately 1 hour, 20 minutes

Lesson 2— Support Hardware and Prototype development tools
Each generic type of IC compatible for use in an 8X300 system configuration is described, using a short form reference to useful devices as an index. Special attention is provided to efficient configuration of working storage and considerations involved in structuring the Interface Vector.

The lesson is concluded with a precise introduction to software support programs and prototype development hardware available to the 8X300 applications designer.

To develop an understanding of support hardware and prototype development tools available for use by the 8X300 system designer.

1. Composite of most useful 8X300-compatible ICs.
2. Explanation:
   82S215 bipolar ROM
   82S115 bipolar PROM
   82S116 bipolar RAM
   8T32/33 IV Byte
3. 256-byte working storage configuration.

4. Expandable IV byte Interface Vector.
5. Introduction to:
   Microcontroller Simulator (MCSIM)
   Microcontroller Cross Assembler Program (MCCAP)
   Designer's Evaluation Board Kit 8T32 Address Programmer

Audio Tape: Approximately 20 minutes
Student
Completion: Approximately 1 hour

Lesson 3—Introduction to Software
This lesson is designed for the programmer who has not previously been exposed to microprocessor software and its relationship to both internal and external hardware associated with the 8X300. An analogy is drawn between software and its relationship to a computer and sheet music as it relates to an orchestra. Technical terms and buzz words related to microprocessors are then introduced and defined.

1. Hardware/software relationships compared with an analogy to a symphonic production.
2. Software technical definitions, symbols, nomenclature.

Audio Tape: Approximately 15 minutes
Student
Completion: Approximately 30 minutes

Lesson 4—8X300 Instruction/Data Formats
This lesson provides basic information on 8X300 instruction and data formats. Instruction classes, including opcode definition and operand introduction, are related to fixed instruction format types. A brief review of number systems, particularly as they relate to the 8X300, is optionally studied by the user.

Outline:

1. General and specific instruction formats.
2. Instruction formats (5) definition.
3. Definition of instruction commands; relationship to formats.
4. Operand introduction—effect on format designation.
5. Data format description.
6. Review of number systems; emphasis on binary and octal (option).

Audio Tape: Approximately 15 minutes
Student
Completion: Approximately 30 minutes

Lesson 5—8X300 Instruction Operands
The 8X300 instruction set is powerfully enhanced by its unique operand configuration. In this lesson, stress is placed on definition and flexibility of all operands as they are written within each instruction class. Using the architecture block diagram, the relationship of each operand to the control of specific internal 8X300 circuits is stressed. Review of all instruction formats is provided, thus laying the groundwork for detailed discussion of the instruction classes themselves.

Outline:

1. Description of all operands (S, D, R/L, I, A) in detail.
2. Software definition of all registers; their features and constraints.
3. Specification of operand control on related hardware.
4. Summary of format/operand relationship.

Audio Tape: Approximately 20 minutes
Student
Completion: Approximately 35 minutes

Lesson 6—8X300 Data Movement Commands
The flexibility of data movement and manipulation are keys to a solid understanding of the 8X300. Within this framework, data movement instructions MOVE and TRANSMIT are exposed. Stress is laid on execution of these instructions with respect to manipulation of data by operand controlled logic distributed within the 8X300 internal data path. Emphasis is placed on dynamic movement of data when registers or selected I/O are designated in any combination as source or destination. Applications and possible software characterization of external hardware are introduced.

Outline:

1. MOVE: Register to Register: Register to I/O; I/O to Register; I/O to I/O. Emphasis on IVL/IVR I/O Port Addressing.
2. TRANSMIT: Immediate Register Load; fixed data to selected I/O. Flexibility and constraints.
3. Summary table of data movement internal controls.

Audio Tape: Approximately 25 minutes
Student
Completion: Approximately 45 minutes

Lesson 7—8X300 Arithmetic/Logic Commands
With prior emphasis on with reference to data movement and manipulation as a base, ALU functions and their resultant effect on data are stressed. Introduction to applications arising from implementation of ALU instructions, and resultant minimization of external hardware required in non-microprocessor systems, is provided. Instruction is provided at the machine language level in order to establish a firm foundation for multi-instruction applications later in the course.

Outline:

1. Arithmetic ADD instruction; function of R-10.
2. LOGICAL AND instruction.
3. EXCLUSIVE OR instruction.
4. Review of ALU and data movement commands through self-administered exercises.

Audio Tape: Approximately 15 minutes
Student
Completion: Approximately 30 to 50 minutes, based on completion of optional review exercises.

Lesson 8—8X300 Program Control Commands
A statement of the purpose of program control instructions is followed by a detailed analysis of each at machine language level. Software partitioning of control storage memory into pages and segments is explained. Emphasis is placed on the dynamics of linking fixed and processed bit patterns into working control storage access addresses. The relationship between absolute memory addresses and immediate operand fields contained within program control instructions is stressed through explanation of, and interactivity with, simplified conversion tables.

Outline:

1. Relationship to 8X300 internal architecture and instruction cycle timing.
2. Memory mapping; software implementation in program control paging, segments, relative displacement.
3. Boundary limits as function of register and selected I/O designation.
4. Solutions for maximizing program control flexibility.
5. Program control examples: XEC, NZT, and JMP.
6. Practical implementation in working program.
7. Description/usage of address/immediate operand translation tables.

Audio Tape: Approximately 35 minutes
Student
Completion: Approximately 1 hour, 10 minutes

Lesson 9—Record Keeping Considerations
This lesson demonstrates the need for effective record keeping as a necessary parallel to multi-instruction software requirements. Top-Down programming concepts are defined. Symbolic labeling of address and data fields are described by purpose and usage. Techniques which must be considered by the programmer in analyzing the need for, and subsequent preparation of, housekeeping routines are proposed. A variety of programming forms, tailored for use by 8X300 software writers, are introduced and explained.

Outline:

1. Need for program enhancement through effective record keeping.
2. Top-Down program design concept introduction.
3. Programmer record maintenance tools and application.
4. CPU record maintenance tools and applications.
5. Record keeping for I/O, process, and address control.
6. Symbolic labeling, usage.
7. Effect of poor record keeping applications on CPU program execution.
8. Techniques for effective record keeping.
9. Programming forms introduction and description.

Lesson 10—Multi-Instruction Applications
The first of two applications lessons describes software sequences for subtracting, inclusive OR, mask variations, basic interrupt processes, CPU and I/O initiated delays, and data deserialization. A number of alternative approaches and solutions to each problem are developed. Transition from single to multiple instruction sequences increases in complexity as the lesson progresses. Housekeeping functions integral to the implementation of each topic are stressed.

Outline:

1. 1's complement, 2's complement, and subtraction.
2. Negative number/immediate operand conversion tables.
3. Software implemented counters, applications.
4. Time delay concepts and applications.
5. Data masking implementation and usage.
6. Masked interrupt process techniques, I/O interrupt handling.
7. INCLUSIVE-OR software and applications.
8. Multiple byte addition and storage features.
9. Serialization and deserialization of data.
   - applications
   - overall program considerations
   - software options
   - significant bit handling
   - interlock with slower I/O
10. Timing and clock generation.
11. Clocked serializing techniques.
12. CPU and I/O controlled delay options including fine tuning techniques.
13. Fixed I/O controlled data serialization.

Audio Tape: Approximately 37 minutes
Student
Completion: Approximately 1 hour 30 minutes

Lesson 11—Multi-Introduction Applications
The second of two applications lessons focuses on multi-bit pattern detection and generation. Applications for data serialization at variably rates determined by I/O. and service routines based on complex I/O status, are described. The subject of interrupt handling is reopened, this time to provide techniques for processing multi-level prioritized interrupts typically found in complex externally controlled I/O systems. Tradeoffs between speed of handling interrupts and conservation of control storage are described, then implemented. Techniques learned in previous examples are then chosen to implement a typical controls application monitoring and service scheme.

Outline:

1. Pattern generation and detection.
2. Cost effective sequence options.
3. XEC instruction flexibility.
4. Multi-line I/O status discrimination techniques.
5. Multiple use of working register; application.
6. Flexible routine exit options.
7. Alternatives:
   Control storage conservation.
   Expected/unexpected scheduled I/O status handling.
8. Multi-level status/interrupt priorities.
9. Software designed for rapid interrupt detection.
10. Complex system's interrupt handling and prioritizing.

Audio Tape: Approximately 35 minutes
Student
Completion: Approximately 1 hour, 20 minutes

Lesson 12—Subroutine Calls & Returns—1
Subroutine calls and returns are examined from 2 basic points of view. These are:

1. Program organization and management.
2. The mechanics of creating effective routines and process sequences.

After a review of program control commands presented in previous lessons, this lesson presents the theory of single and multi-level routine access with sequentially increased complexity. Possible programmer errors are defined and practical solutions for their prevention are suggested. Tradeoffs in memory conservation, speed, and dynamic operations related to program organization and execution are examined. 8X300 program control instruction flexibility is stressed.

Outline:

1. Introduction: The need for effective overall program organization and management.
2. Review of elementary intra-routine access procedures.
3. 8X300-oriented return address link:
   Basic use of working register.
   Return link saved in working storage; release of register. Return jump table usage; Memory mapping.
4. Routine nesting; concept, need, principles, considerations.
5. Practical two-level routine access and return.
6. Simplified multi-level subroutine/return structure.
7. Legal and Illadvised sequences; constraints; options to eliminate error.
8. Page and segment boundary management.
9. Diagrammed analysis of nesting multi-level program structures.
10. Practical choice of all/return instruction procedures based on program segment definition:
    Terminal
    Inline
    Bypass
    Multiple access
11. Summary of fundamental program structuring principles.

Audio Tape: Approximately 37 minutes
Student
Completion: Approximately 1 hour, 10 minutes

Lesson 13—Subroutine calls and Returns-2
Implementation of subroutine multi-level access control through software configuration of the RETURN ADDRESS STACK (RAS) is presented. Different types of RAS are defined with detailed exposition of one easily adapted to 8X300 usage. Configuration of the RAS in working storage and accompanying universal jump table access is presented. The use of intermediate software processes for dynamic access of, and return from, nested program segments is described. Simplification of program complexity is stressed.

Program management considerations are introduced, including subtleties involved in program packing for control storage conservation and error correction involving page and segment boundary management.

Outline:

1. RETURN ADDRESS STACK definition and architecture.
2. Software generation of RAS:
   Usage of working storage
   Universal Jump Table
3. Direct versus indirect subroutine access/return.
4. Implementation of indirect processes details, constraints, options.
5. Relocatable addressing in multi-level sequencing.
6. Program packing; advantages and need.
7. Page and segment boundary management errors and solutions.

Audio Tape: Approximately 35 minutes
Student
Completion: Approximately 1 hour, 10 minutes

DESCRIPTION

The 8X300 Assembler is a program which accepts source code written in the 8X300 assembly language and which produces both a listing of the symbolic program and an object module in one of three formats: MCSIM, ROM Simulator or BPNF.

The assembler is written in ANSI standard Fortran and is approximately 2800 card images in length. It requires some rewindable I/O medium such as disk or tape for temporary storage, an input device to read source code and two output devices, one to output the assembler listing and one to output the object module. If desired, the assembler may be compiled and linked to execute in overlays.

The assembler consists of two passes which build a symbol table, issue helpful error messages, produce an easily readable program listing and output a computer readable object module.

FEATURES

- **Macros nested to three levels**
- **Conditional assembly**
- **Address arithmetic**
- **Automatic procedure/subroutine handling**
- **Reserved symbols for registers**
- **Multiple entry to procedures**
- **Symbolic machine operation codes**
- **Symbolic address assignment references**
- **Symbolic data creation statements (IV Byte References)**
- **Free format source code**
- **Syntax error checking**
- **Self-defining constants**
- **Assembly listing control statements**
- **ASCII character code**
- **Comment statements and comment areas**
- **Forward referencing**

MCCAP LANGUAGE

The assembler recognizes three types of statements: comment statements, machine instructions, and pseudo-ops (directives).

Comment statements are used to document a listing. They must contain an asterisk in column 1 and may contain any legal character in the other 79 columns.

Machine instruction statements are those statements that generate instruction(s) to be executed by the 8X300 processor. Machine instruction statements include:

MOVE	Move byte from one location to another.
ADD	Add the byte in one location to the byte in the auxiliary register and store the result in another location.
AND	AND the byte in one location with the byte in the auxiliary register and store the result in another location.
XOR	Exclusive OR the byte in one location with the byte in the auxiliary register and store the result in another location.
XEC	Execute the instruction at the specified address.
XMIT	Transmit the literal (immediate data) contained in this instruction to the specified location.
NZT	Jump to the specified address if the value in the specified location is not zero.
JMP	Jump to the specified address.
CALL	Transfer control to a procedure or an entry point.
RTN	Returns control from a procedure or an entry point.
SEL	Places the address of an IV Byte variable into the IVL or the IVR register.

Pseudo-Ops (or Directives) are commands to the assembler, but have no meaning to the 8X300 processor.

EQU	Assigns a value to the symbol in the label field.
SET	Identical to EQU, except that the symbol may be redefined by another SET statement.
LIV	Assigns a symbolic name to the address, position and precision of the "left bank" Interface Vector Byte variable.
RIV	Identical to LIV, except that the assignment is made to the "right bank" I/O Field.
PROG	Introduces and names the main program.
PROC	Introduces and names a procedure.
ENTRY	Specifies an additional entry point to a procedure.
END	Terminates a procedure, a program or a MACRO.
ORG	Changes the value of the location counter.
OBJ	Specifies the format of the object module.
IF	Introduces source statements subject to conditional assembly.
ENDIF	Terminates the source statements subject to conditional assembly.
LIST	Specifies the list and punch options.
NLIST	Negates the list and punch options.
EJCT	Advances listing to next page.
SPAC	Advances listing to next line.
MACRO	Defines the source statements generated a macro call.

AVAILABILITY

The assembler is available on NCSS, GE and TYMSHARE timesharing services. It is also available from Signetics on 9-track magnetic tape written in EBCDIC in 80-character unblocked records at a density of 800 bpi.

FEATURES

- **Totally self-contained with keyboard alpha-numeric display, tape reader, TTY output**
- **Dual MicroControllers: one to run instrument, one dedicated to execute user's program**
- **Real time instruction execution**
- **Control of program execution-Halt Single Step and Run Modes**
- **Direct program/register/IV examination and modification through keyboard**
- **Three breakpoint types—Normal, Halt and Insert Instruction**
- **Up to 4k words of high speed read/write program storage**
- **Format compatible with papertape output of MicroController Cross Assembly Program**

DESCRIPTION

The SMS MicroController Simulator, MCSIM, is a hardware development instrument designed to perform in the user's system exactly as an SMS MicroController. It directly supports the SMS 300 (8X300) as well as MicroController prototyping systems. MCSIM gives the user an interpreter system with a modifiable Program Storage and a Control Panel which together provide a means of entering, running, monitoring, debugging and changing a program. It features ease of use due to its high level interactive display/keyboard and simple operating system. MCSIM allows the user to run his program on-line in real time. Through the Control Panel, data stored in internal registers, Working Storage and IV Bytes can be examined and modified. An extensive breakpoint capability permits the user to quickly locate program faults.

MCSIM contains two MicroControllers: one for running the instrument and a second totally dedicated to the user's program and prototyping system. This insures that when program development is complete and the design specifications have been met, production systems will perform identically with the prototype.

OPERATION

MCSIM operation is separated into six modes, each mode consisting of a related set of functions and status displays. Access to a mode is made using one of the six mode selection keys. (See Table 1). The currently selected mode is displayed at the extreme left part of the display; the currently selected function is displayed at the extreme right of the display. Selection among the available functions is made using the ↑ and ↓ keys, incrementing and decrementing to the next function. The function selected at last exit is automatically re-selected when the mode is re-entered except for the Breakpoint Mode which selects the Current display.

An operation is generally performed using the following four steps:

1. Select a mode by depressing one of the six Mode Select Keys.
2. Select a function in that mode by positioning the function roll with one of the two Function Select Keys.
3. Set up any necessary conditions, such as entering data or readying papertape in the reader.
4. Activate the function by depressing the Function Acknowledge Key (FCN/ACK).

MODE	FUNCTIONS
Manual Examination and alteration of Program Storage	Change Address Store Instruction
Tape Load or dump all or part of Program Storage	Load Verify Begin Punch Address End Punch Address Punch Program Identification
Register Examination and alteration of the Interpreter's internal registers	Change Address Store Octal Data Store Binary Data Complement Overflow
Interface Vector Examination and alteration of the locations on the IV Bus	Display Current Enabled Bytes Change IV Address Store Binary Data in IV Location
Breakpoint Set one of three types of breakpoint for program debugging	Display Currently Active Breakpoint Set Normal (Sync) Breakpoint Set Stop Breakpoint Replace Instruction at Breakpoint
Execute Control and monitor execution of a program	Halt Run Program Insert Instruction Single Step

Table 1 MCSIM OPERATOR CONTROLLED FUNCTIONS

SPECIFICATIONS

Control Panel

The Mode Select Keys and the Function Select Keys used in combination give the user 26 possible functions to accomplish complete loading and testing of the program. Each Mode Select Key accesses a set of functions, one of which is chosen using the Function Select Keys.

Status Messages

Message displays are used to indicate special conditions which may result from the operation of MCSIM:

POWER ON, MCSIM SYSTEM START
???MORE THAN ONE KEY PRESSED
UNABLE TO READ TAPE. RESTART
UNDEFINED MEMORY ADDRESS (octal address)
INVALID INSTRUCTION (instruction code)
LOADING INTERRUPTED, RESTART
V'FYING INTERRUPTED, RESTART
P'CHING INTERRUPTED, RESTART
NO VERIFY (octal address, tape data, memory data)
CLEAR RESET LINE TO START RUNNING
CLEAR HALT LINE TO START RUNNING
INVALID 8-BIT OCTAL VALUE (octal value)
UNDEFINED IV BYTE ADDRESS (octal address)

Sync Output

Output pulse whenever address breakpoint is reached.

System Interface

MCSIM is connected to the user's prototyping system via a single ribbon cable. This cable terminates in either a MicroController Simulation Module or a dual in-line plug which is pin for pin compatible with the SMS 300 Interpreter. There are two different simulation modules to exactly match presently available MicroController systems. Selection of the proper input/output connector makes a MCSIM appear to be physically and electrically equivalent to a production MicroController or Interpreter.

Data Input/Output

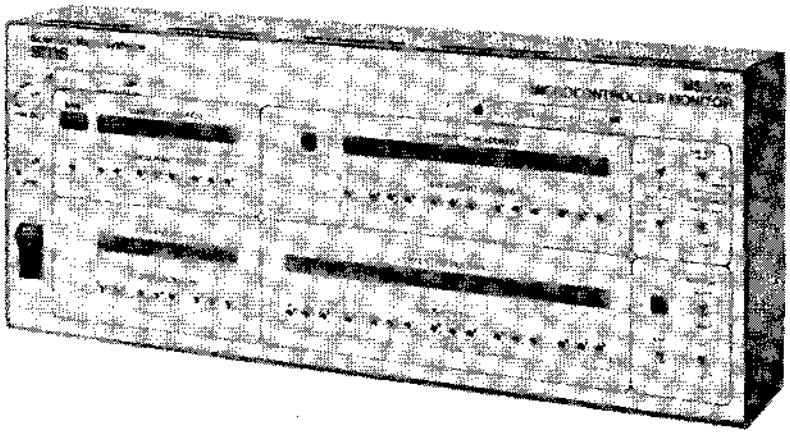
COMPONENTS	DESCRIPTION
Panel	36 character self scan display for messages and data readout 12 key numeric keyboard for data entry/modification
Tape Reader	120 character/second tape reader for high speed data entry (ASCII format)
TTY	20mA current loop output for listing of program code on Teletype® terminal

PHYSICAL CHARACTERISTICS

Power	115V or 230V ± 10% 50 or 60Hz ± 10% 350 watts/min. to 750 watts/max. (Power dissipation dependent on the number of Simulation Modules configured in the system)
Dimensions	7 inches high x 17 inches wide x 19 inches deep
Installation	May be used on table or may be installed in standard 19 inch rack with mount adapters
Weight	65 lbs. net, 75 lbs. shipping
Ventilation	Air Flow 120 CFM
Environmental	0° to 55°C, Relative Humidity to 90%.

FEATURES

- **Real time monitoring instrument for SMS 300**
- **Totally self contained**
- **Displays IV address and data**
- **Displays current program address**
- **Displays current instruction**
- **Control of RESET and HALT**
- **Single step capability**
- **Real time instruction insertion**
- **Two real time breakpoints**
- **Breakpoint output signal**

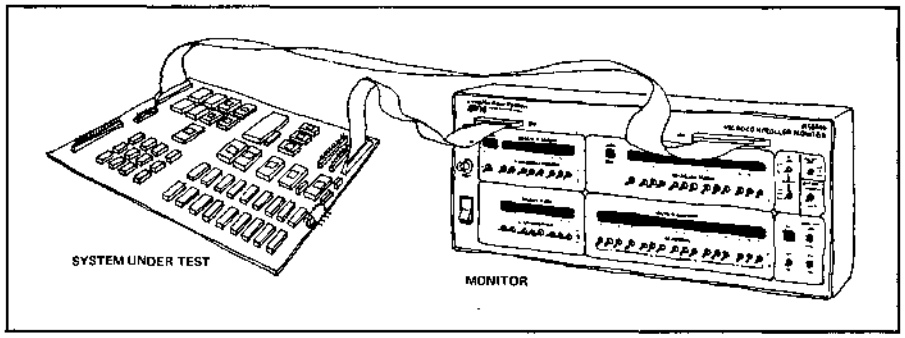


DESCRIPTION

The MicroController Monitor is a self contained debug and maintenance tool for use with all systems containing the SMS 300 (8X300). It provides a control panel allowing an operator to observe, modify, and control program execution in real time permitting system faults to be rapidly traced. The Monitor displays the status of the Address, Instruction, and IV Bus of an Interpreter. Switch registers associated with each display can be used to set breakpoint addresses and enter instructions or data from the Monitor. Program execution can be halted, single stepped, or altered from Monitor controls.

The Monitor can be used to insert instructions thereby examining or modifying the contents of internal registers, Working Store, and IV Bytes. In addition, it can be used to start program execution from any address and enable the operator to set program loops allowing a program segment to be checked independently of normal program flow. Breakpoints on IV data and program address allow the operator to halt program execution or insert instructions in real time.

As shown in the diagram below, the Monitor connects to the system under test through flat ribbon cables (provided) or an adaptor (optional) which plugs into an Interpreter socket. These connections are buffered and present a negligible load to the system. Other features include a Sync output which provides pulses at a selectable program address or Interface Vector event.



OPERATION

The MicroController Monitor provides the user with two basic modes of system operation, RUN and STEP. When in the RUN mode, indicator LED's provide a continuous real time display of the three major system busses: Address, Instruction, and Data. Program execution can be halted by one of two actions: depressing the Halt switch or selecting the Halt on Breakpoint function. When halted, it is possible to execute one instruction at a time by pressing the RUN/STEP control to the STEP position. The readout shows the current (static) bus information and the next instruction to be executed.

For checkout purposes it is useful to be able to set a program loop or modify the normal progam flow in some other manner. This is accomplished by inserting in real time the instruction set up on the Instruction Switch Register whenever the Program Address Breakpoint is reached. To check for system noise or similar malfunction a separate Insert Instruction switch can be set to the Multiple position, forcing one instruction repetitively. When halted, a single instruction can be inserted by toggling the Insert Instruction switch to the Single position. When an instruction is being inserted, system ROM is temporarily disabled.

To aid in the diagnosis of a malfunction an advanced breakpoint capability is provided. In addition to the Halt and Insert functions when a Program Address Breakpoint is reached, a breakpoint can be generated on the Interface Vector (IV) Bus data or address. This permits program execution to be halted or a Sync pulse generated in response to external data or the contents of RAM. Three modes of IV Breakpoint generation are selectable: Breakpoint on IV Address; Breakpoint on IV Address and IV Write Data; and Breakpoint on IV Address and IV Read Data. The IV Breakpoint Data switches have a "don't care" position to permit generation of a breakpoint on a data subfield. A Sync pulse output is provided on either the IV Breakpoint or Program Address Breakpoint (switch selectable).

SPECIFICATIONS

Controls	Function
RUN/STEP	Sets operation mode, continuous execution (RUN) or single step (STEP).
INSERT INST	Unconditionally causes execution of the instruction in the Instruction Switch Register.
RESET/HALT	Three position switch used to unconditionally HALT program execution or RESET Interpreter's Program Counter to zero.
PGM ADR BKPT	Selects appropriate function to be performed when Program Address Breakpoint is reached: HALT—halt on breakpoint, INSERT—replace normal instruction with instruction set into Instruction Switch Register, OFF—program execution unchanged by breakpoint.
IV BKPT	Selects appropriate function to be performed when IV Breakpoint is reached: HALT—halt on breakpoint, OFF—program execution unchanged by breakpoint.
BKPT UPON	Selects IV Breakpoint on one of three conditions: specified IV address, specified write data at IV address, specified read data at the IV address.
SYNC SOURCE	Selects source for output Sync pulse, IV Breakpoint or Program Address Breakpoint. Sync pulse always available at breakpoint regardless of function selected by breakpoint controls.
Switch Registers	
INSTRUCTION	Sixteen two position switches for entering instruction to be inserted (during breakpoint or insert instruction control).
BREAKPOINT ADDRESS	Thirteen two position switches to select program breakpoint address.
IV BREAKPOINT ADDRESS	Eight two position switches to select IV byte breakpoint address or Working Store breakpoint address.
BANK	Selects display of LB (Left Bank) or RB (Right Bank) IV address information. Also selects either LB or RB for IV breakpoint.
IV BREAKPOINT DATA	Eight three position switches, 0, 1, X (don't care), to select IV Data Breakpoint.
Displays	
RUN	LED display indicating when Interpreter not halted.
CURRENT INSTRUCTION	LED display of the binary value of next instruction to be executed.
CURRENT PROGRAM ADDRESS	LED display of the binary address of the next instruction to be performed.
IN XEC RANGE	LED display indicating that the value of the Interpreter's program counter and address register may not match because the previous instruction command was "Execute."
BANK	Displays band addressed by the current instruction (left bank or right bank).
CURRENT IV ADDRESS	Shows the binary address of the currently enabled IV Byte or Working Store location.
CURRENT IV DATA	Binary display of the data on the IV bus.
Outputs	
SYNC	Pulse output whenever the switch selected breakpoint occurs.
PGM EN(L)	A low true signal used to enable or disable system program ROM (required only when Interpreter adaptor is used).
Cycle Time	
The Monitor can be adjusted to work with any MicroController-based system with a cycle time of 200ns to 2μs.	

PHYSICAL CHARACTERISTICS

Power	115Vac ± 10% at 0.75 amp max. 230Vac ± 10% at 0.38 amp max. (optional) 47-63Hz
Dimensions	7" H x 17.5" W x 3" D
Installation	May be used on table or may be installed in standard 19 inch rack (using optional adaptors).
Weight	10 lbs. net, 13 lbs. shipping
Ventilation	Forced air, 120 CFM.
Environmental	0° to+50°C, relative humidity to 90%
Cables	2 provided, 3 feet long

APPLICATION MEMOS

DESCRIPTION

Typical interfaces to the 8X300 employ the 8T32/33 or 8T35/36 bidirectional I/O ports. These devices provide a single connnection between the 8X300 and the user status control and data lines. Each interface is denoted as an Interface Vector and is field programmed to a specific address.

ADDRESSING DATA ON THE INTERFACE VECTOR

The Interface Vector is comprised of general purpose 8-bit I/O registers called Interface Vector (IV) Bytes. The IV registers serve to select IV bytes. In order for an instruction to access (read or write) an IV byte, the address of that byte must be output to the IVL or IVR registers.

Thus, two instructions are required to operate on an Interface Vector byte:

XMIT ADDRESS, IVL MACHINE INSTRUCTION

Each of the two IV registers (IVL and IVR) may be set to select an IV byte, therefore two I/O ports may be active at one time—one on the Right Bank (IVR) and one on the Left Bank (IVL). Data may be input and output in one instruction following the selection of IV bytes:

```
XMIT    ADDRESS1,IVL
XMIT    ADDRESS2,IVR
ADD     LB, RB
```

Once the IV byte is selected (addressed) it will remain selected until another address is output to the same IV register. Since an IV register (IVL, IVR) can be used only as a destination field of an instruction, any instruction sending data to IVL or IVR can be used to select an IV byte.

From the user's standpoint, however, all IV byte outputs can be read by an external device regardless of whether they are selected or not.

The address range of IVL and IVR is $0\text{-}255_{10}$.

ELECTRICAL CHARACTERISTICS OF THE INTERFACE VECTOR

Each IV byte consists of 8 storage latches which hold data transferred between the Interpreter and the User System, 8 tri-state input/output lines and 2 input/output control lines, called Byte Input Control (BIC) and Byte Output Control (BOC) as shown in Figure 1. The control lines functions are summarized in Table 1. Table 2 contains a summary of the electrical characteristics of the IV byte.

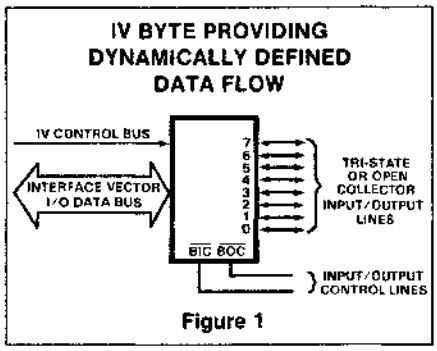


Figure 1

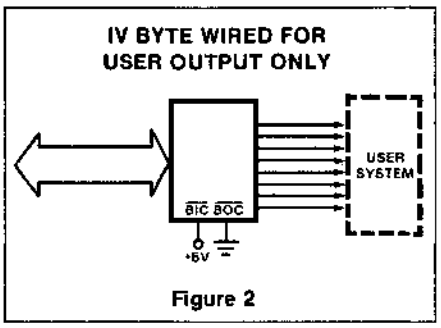


Figure 2

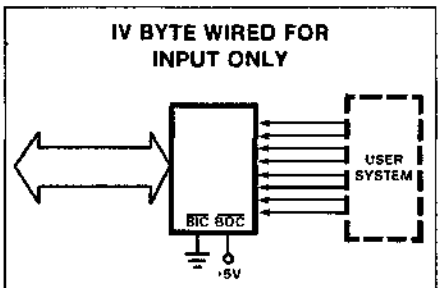


Figure 3

Working storage consisting of RAM may be connected to either or both left and right I/O banks. An example of such an arrangement is shown in Figure 4. Paging may be added to the memory to extend the addressability.

CONTROL LINES		FUNCTION
BOC (low true)	BIC (low true)	
H	H	8 I/O lines in high impedance state—disable
L	H	8 I/O lines in output mode—8 bit storage latch data available in the output lines.
X	L	8 I/O lines in input mode—data can be read by Interpreter

Table 1 FUNCTIONS OF THE BIC AND BOC LINES

PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
V_{IH}	High level input voltage		2		5.5	V
V_{IL}	Low level input voltage		-1		0.8	V
V_{IC}	Input clamp voltage	I_{IL} = -5mA			-1	V
V_{OH}	High level output voltage	I_{OH} = 1mA	2.4			V
V_{OL}	Low level output voltage	I_{OL} = -16mA			0.5	V
I_{IH}	High level input current[1]	V_{IH} = 5.5V			100	uA
I_{IL}	Low level input current[1]	V_{IL} = 0.50V			-800	uA
I_{OS}	Output short circuit current	V_{OL} = 0V	-20		-200	mA
C_{IN}	Data input capacitance	V_{IL} = 0V			12	pF

NOTE

1. Input current is always present regardless of the state of BIC and BOC.

Table 2 IV BYTE TERMINAL ELECTRICAL CHARACTERISTICS

8X300 TYPICAL SYSTEM CONFIGURATION
WITH WORKING STORAGE

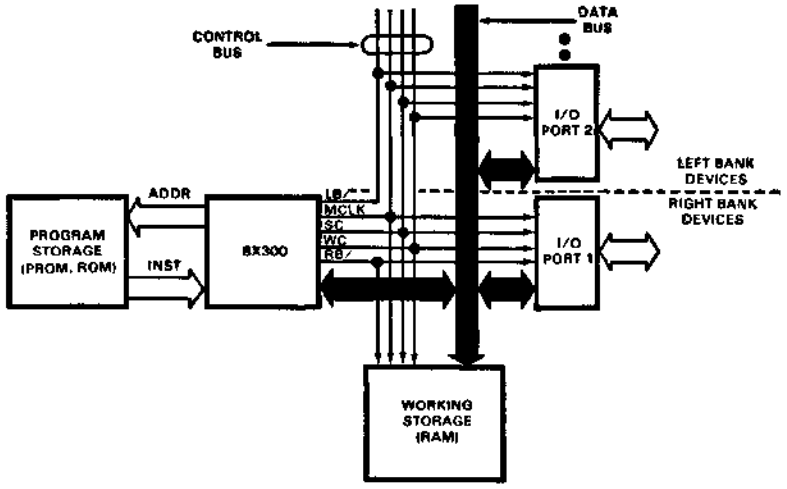


Figure 4

FLOPPY DISC INTERFACE

DESCRIPTION

The 8X300 controls a floppy disc drive with a minimal amount of additional circuitry. In this example, byte assembly and disassembly are performed by the program ("bit banging") to reduce interface circuitry. Addition of such circuitry would increase hardware costs and decrease significantly peak processor utilization.

Data is transferred to and from the floppy disc via I/O driver routines. These I/O driver routines provide a standard software interface to a floppy disc and require 180 words of program storage. When not transferring data to and from the disc, the 8X300 is available to service other devices such as keyboards, displays or data communication lines. Figure 1 illustrates the system.

DESIGN APPROACH

Data bytes are assembled or disassembled by sensing a clock, inputing data bit or generating clock, and outputing a data bit. Preamble patterns, track address, and other disc format requirements are implemented by programming. Disc drive head must be stepped to the desired track before data transfer is initiated. Disc drive status is monitored to determine any error conditions. A four step procedure is followed to implement the design:

1. Analyze interface. Analyze floppy disc relative to: Number of control/data lines, timing and data rates associated with each control/data line, and electrical characteristics of each control/data line. Determine any supplemental circuits needed for electrical compatibility (see Table 1).
2. Perform functional analysis. The functions to be programmed and any which require supplemental logic are determined. In this case, supplemental logic is utilized to process the 250ns pulses associated with DATA, CLOCK and WR DATA. Optional logic for byte assembly and disassembly also are shown. The programmed functions are:
   a. Byte assembly/disassembly
   b. Generate preamble, track address, timing and sector synchronization
   c. Sense clock and disc status
   d. Step head to desired track
3. Define the program to process input and to generate output (see Figure 2).
4. Determine 8X300 configuration (see Table 2).

FLOPPY DISC INTERFACE

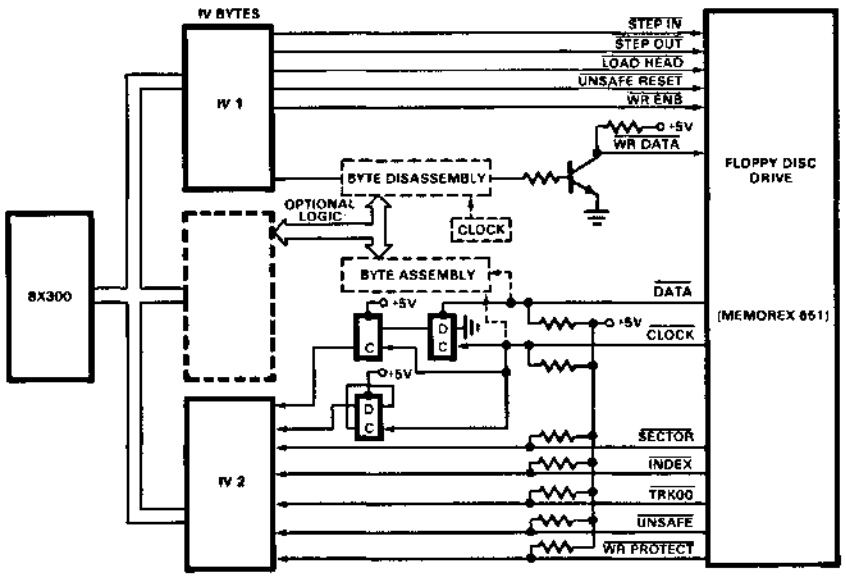


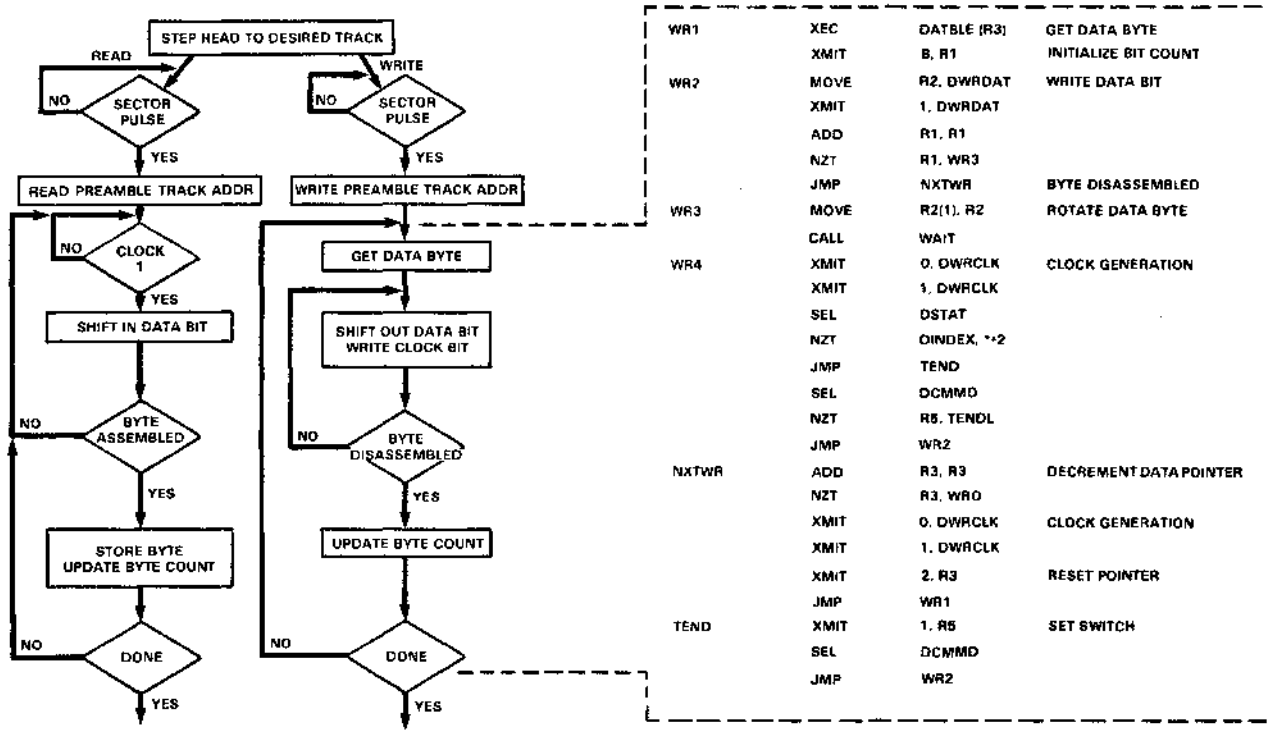
Figure 1

SIGNAL NAME	DATA RATE[1]	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
STEP IN	20ms	.01-10ms	TTL with pullup	1		Step head 1 track in
STEP OUT	20ms	.01-10ms	TTL with pullup	1		Step head 1 track out
LOAD HEAD	Level		TTL with pullup	1		Load head
UNSAFE RESET	Level		TTL with pullup	1		Clears unsafe condition
WR ENB	Level		TTL with pullup	1		Enables write operation
WR DATA	2μs	.25μs	50mA current	1	2R,T	Data/clock to disc
SECTOR	5ms	1ms	OC output	1	R	Sector indicated
INDEX	160ms	1ms	OC output	1	R	Begin of track indicator
TRK00	Level		OC output	1	R	Head on track 00
UNSAFE	Level		OC output	1	R	Unsafe condition indicator
WR PROTECT	Level		OC output	1	R	Write protected disc
DATA	4μs	.25μs	OC output	1	R,2FF	Data from disc
CLOCK	4μs	.25μs	OC output	1	R,FF	Clock from disc

R = Resistor
T = Transistor
FF = Flip-Flop

Table 1 INTERFACE ANALYSIS

FUNCTIONAL ANALYSIS



Through put: (a) Peak data rate: 250K bits/sec. (b) Peak processor utilization: 97.5%, including byte assembly/disassembly. (c) Peak processor utilization: 12.2%, with external byte assembly/disassembly.

Figure 2

ROM/PROM FOR PROGRAM STORAGE		WORKING STORAGE FOR DATA BUFFERS	IV BYTES FOR INPUT/OUTPUT INTERFACE
Input Driver	76 words	256 Bytes	6 IV bits for output
Output Driver	74 words		7 IV bits for input
Head Step Driver	30 words		Total: 2 IV bytes
Total	180 words		

Table 2 8X300 CONFIGURATION

TELETYPE MULTIPLEXER

DESCRIPTION

The 8X300 is easily interfaced to a teletype or similar asynchronous device. Processor utilization is less than .1%, even when used in a character assembly mode.

A single 8X300 can be used as a multiplexer for many low speed asynchronous devices. For example, the 8X300 can be used as a front end multiplexer for a large computer system. Figure 3 illustrates the system.

DESIGN APPROACH

A basic teletype I/O driver routine receives, transmits and echoes a character. Character assembly/disassembly is implemented by sensing start bit, sampling data bit and generating output bit timing. A four-step procedure is followed to implement the design:

1. Analyze interface. Analyze teletype relative to: Number of control/data lines, timing and data rates associated with each control/data line, electrical characteristics of each control/data line, and determine any supplemental circuits needed for electrical compatibility (see Table 3).
2. Perform functional analysis. The functions to be programmed and any which require supplemental logic are determined. In this case, no supplemental logic is required. The programmed functions are:
   a. Character assembly/disassembly
   b. Sense start bit
   c. Generate bit timing and simultaneous character echo
3. Define the program to process input and to generate output (see Figure 4).
4. Determine 8X300 configuration (see Table 4).

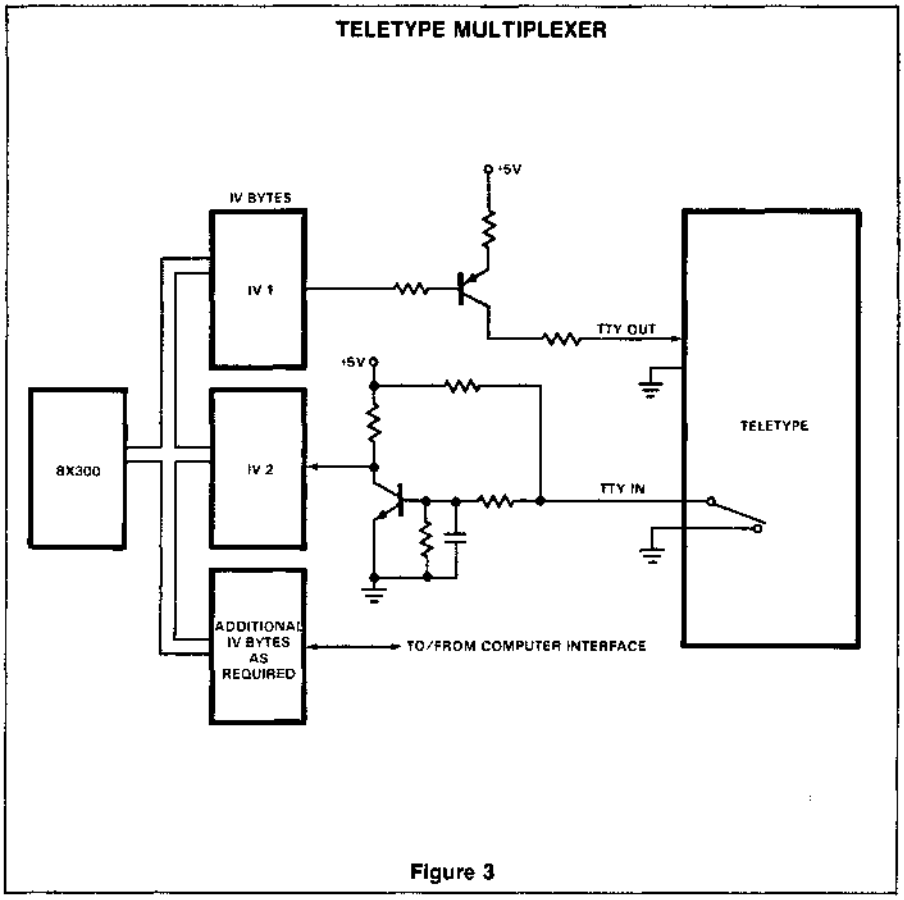


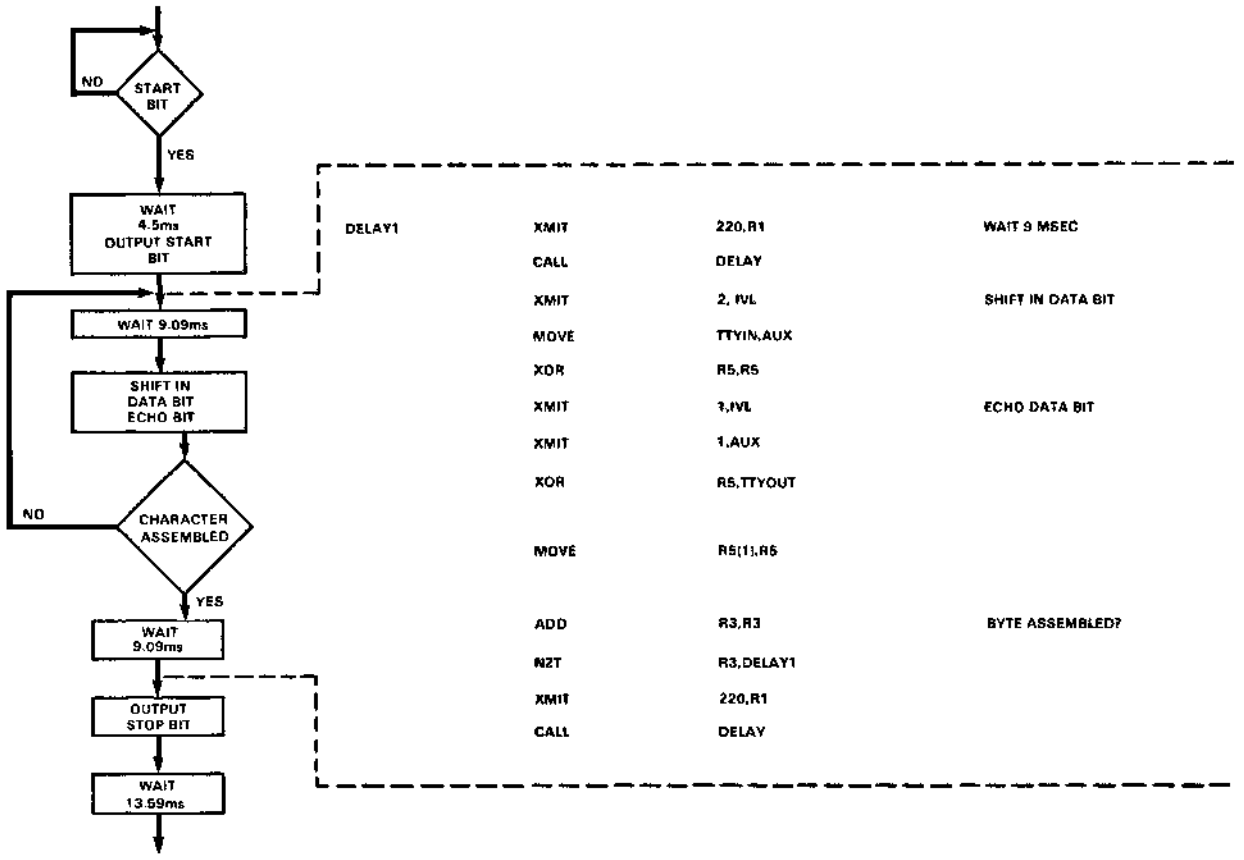
Figure 3

SIGNAL NAME	DATA RATE	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
TTY OUT	9.09ms	9.09ms	20mA current	1	3R,T	Data to TTY printer
TTY IN	9.09ms	9.09ms	20mA current	1	4R,T,C	Data from TTY keyboard

R = Resistor
T = Transistor
C = Capacitor

Table 3 INTERFACE ANALYSIS

FUNCTIONAL ANALYSIS



Through put: (a) Peak data rate: 220 bits/sec. (b) Peak Processor utilization: .05%

Figure 4

ROM/PROM FOR PROGRAM STORAGE	WORKING STORAGE FOR DATA BUFFERS	IV BYTES FOR INPUT/OUTPUT INTERFACE
Teletype driver ..... 49 words Delay routine ..... 10 words Total ..... 59 words	2 bytes per Teletype	1 IV bit, for output, per Teletype 1 IV bit for input, per Teletype Total: 2 IV bytes per 8 Teletypes

Table 4 8X300 CONFIGURATION

DATA CONCENTRATOR

DESCRIPTION

The 8X300 multiplexes multiple low speed terminals. It buffers the data in its working storage for efficient transmission over common carrier or other data link facilities. Single inquiry/response terminals are interfaced to a single half-duplex synchronous line via a Universal Asynchronous Receive-/Transmit (UART) interface. This eliminates cabling to each terminal. The 8X300 transfers inquiry and response messages between terminals and a remote computer data base via a data communications line. Various communication data rates are accommodated by simple program modification. Figure 5 illustrates the system.

DESIGN APPROACH

The 8X300 polls each terminal requesting an input character or signaling an output character. Each character is transferred over a high speed (9600 baud) synchronous line whose data rate determines the scan time of the 8X300 multiplexing program. The multiplexer program formats polling messages, maintains status, generates and checks the Longitudinal Redundancy Character, performs character recognition, and buffers characters. Additional driver programs are required to communicate with the full duplex data communications line to/from a remote computer data base. A four step procedure is followed to implement the design:

1. Analyze interface. Analyze UART relative to: Number of control/data lines, timing and data rates associated with each control/data line, electrical characteristics of each control/data line and determine any supplemental circuits needed for electrical compatibility (see Table 5).
2. Perform functional analysis. The functions to be programmed and any which require supplemental logic are determined. In this case, no supplemental logic is required. The programmed functions are:
   a. Maintain current line status
   b. Generate synchronization pattern, poll command, sense character synch
   c. Resynchronize with clock and monitor modem and UART status
3. Define the program to process input and to generate output (see Figure 6).
4. Determine the 8X300 configuration (see Table 6).

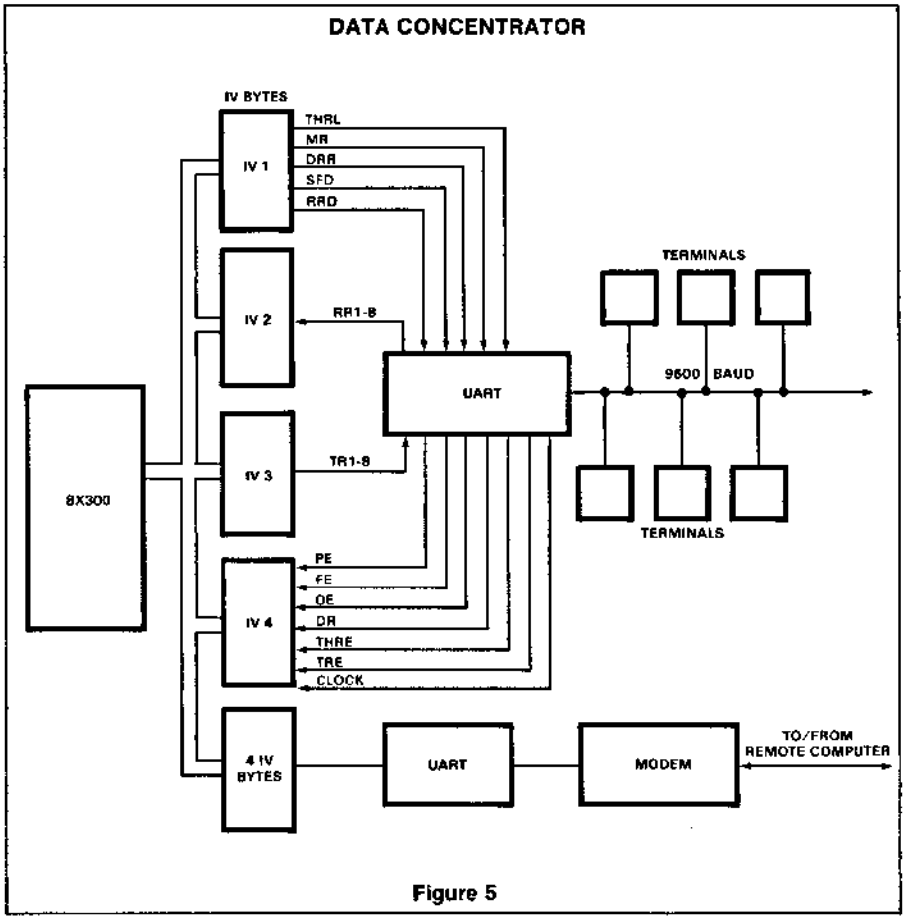
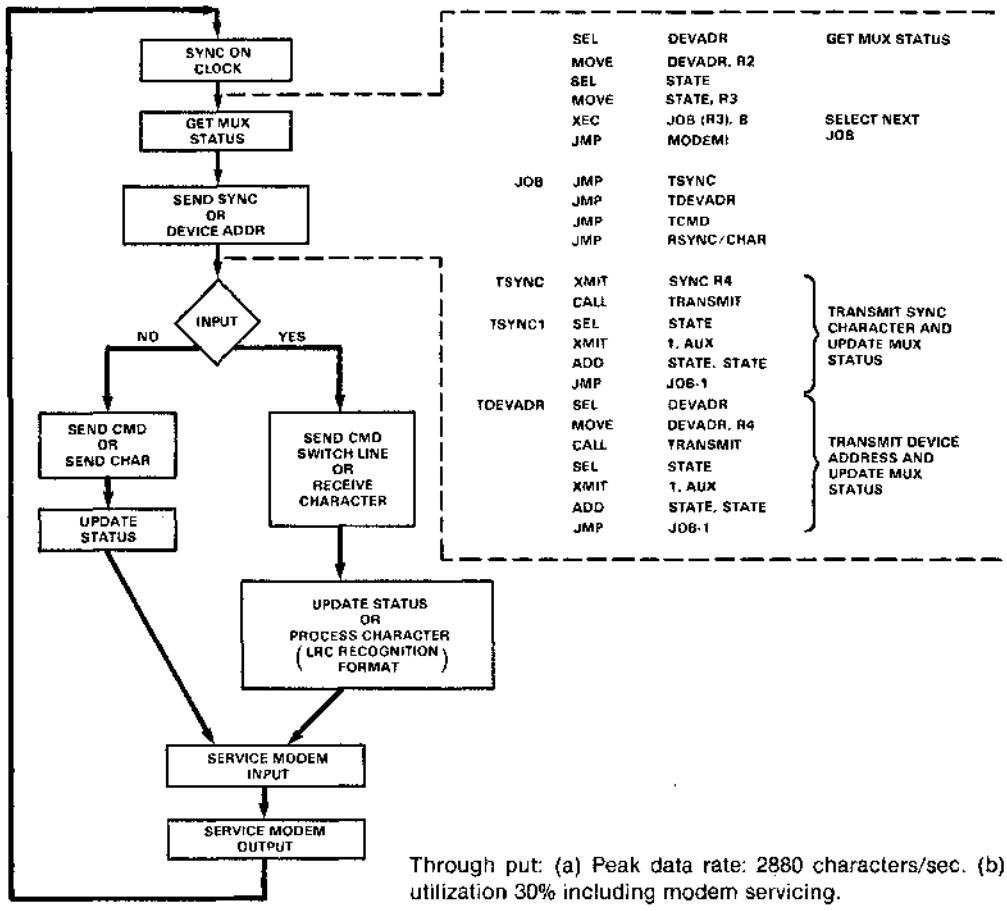


Figure 5

SIGNAL NAME	DATA RATE	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
TR1-8	1.041ms	1.2-10μs	TTL	8	-	Output data
THRL	1.041ms	1.2-10μs	TTL	1	-	Load output data
MR	level		TTL	1	-	Master reset
DRR	level		TTL	1	-	Data received reset
SFD	level		TTL	1	-	Status flag disable
RRD	level		TTL	1	-	Receiver Register disable
RR1-8	1.041ms	1.041ms	TTL	8	-	Received data
PE	level		TTL	1	-	Parity error
FE	level		TTL	1	-	Frame error
OE	level		TTL	1	-	Over run error
DR	level		TTL	1	-	Data received flag
THRE	level		TTL	1	-	XMTR holding reg. empty
TRE	level		TTL	1	-	Transmitter register empty
CLOCK	1.041ms	1.041ms	TTL	1	-	Data rate clock

Table 5 INTERFACE ANALYSIS

FUNCTIONAL ANALYSIS



Through put: (a) Peak data rate: 2880 characters/sec. (b) Peak processor utilization 30% including modem servicing.

Figure 6

ROM/PROM FOR PROGRAM STORAGE	WORKING STORAGE FOR DATA BUFFERS	IV BYTES FOR INPUT/OUTPUT INTERFACE
Multiplexer driver ..... 156 words Character processing ..... 100 words Total ..... 256 words	32 bytes	13 IV bits for output per UART 15 IV bits for input per UART Total: 4 IV bytes per UART

Table 6 8X300 CONFIGURATION

REMOTE ALPHANUMERIC TERMINAL CONTROLLER

DESCRIPTION

The 8X300 interfaces to simple keyboard/display devices with a minimal amount of interface circuitry. The display may be buffered or the 8X300 system can supply buffering and refresh. In this example, the personality of the keyboard/display terminal is programmed into program storage to implement various editing and format functions. A single 8X300 can be used to control a local cluster of alphanumeric terminals since the processor utilization for a single terminal is very low. Messages to and from each terminal are transferred to a remote computer (interface not shown). Figure 7 illustrates the system.

DESIGN APPROACH

A terminal driver routing inputs and buffers messages in working storage. The driver also performs character and line deletion functions and implements a flicker free display of the message. A special set of control characters are used to terminate a message and forward the message. A four step procedure is followed to implement the design:

1. Analyze interface. Analyze keyboard and display relative to: Number of control and data lines, timing and data rates associated with each control/data line, electrical characteristics of each control/data line and determine supplemental circuits needed for electrical compatibility. Here the interfaces are completely compatible electrically (see Table 7).
2. Perform function analysis. The functions to be programmed and any which require supplemental logic are determined. In this case, no supplemental logic is required. The programmed functions are:
   a. Store a message input from keyboard
   b. Update display to produce flicker free output
   c. Implement character delete, line delete editing functions
   d. Recognize end of message control character.
3. Define the program to process input and to generate output (see Figure 8).
4. Determine the 8X300 configuration (see Table 8).

REMOTE ALPHANUMERIC TERMINAL COUNTER

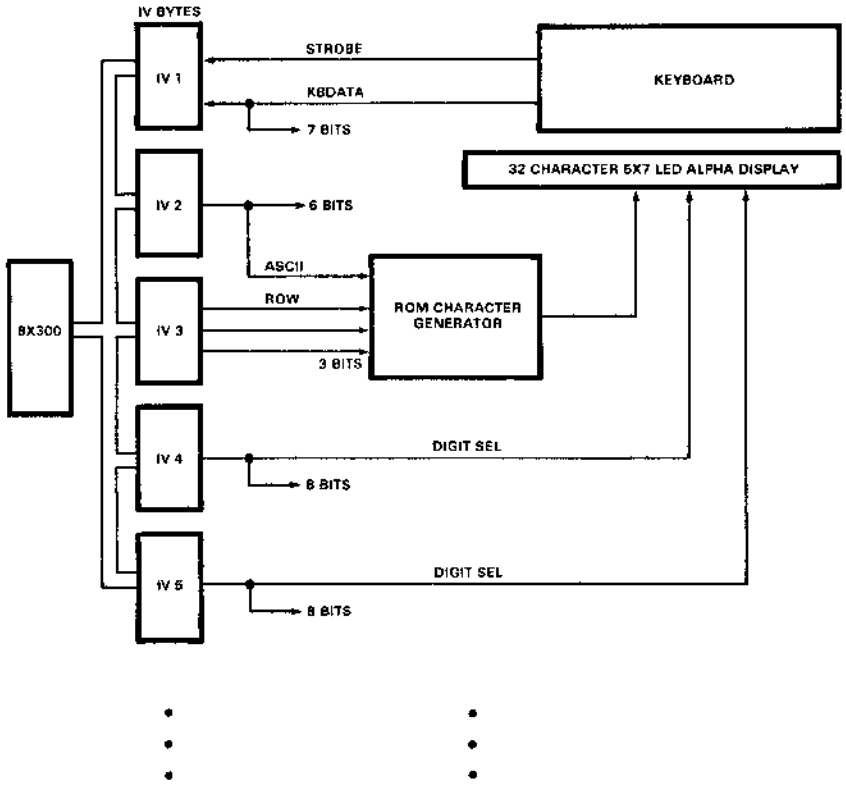


Figure 7

SIGNAL NAME	DATA RATE	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
STROBE	level	4msec (min)	TTL	1	-	Input Character ready
KBDATA	level	4msec (min)	TTL	7	-	Keyboard input character
ASCII	level	16.6msec (max)	TTL	6	-	Select character
ROW	level	16.6msec (max)	TTL	3	-	Select row of digit
DIGIT SEL	level	200ns (min)	TTL	32	-	Select digit for display

Table 7 INTERFACE ANALYSIS

FUNCTIONAL ANALYSIS

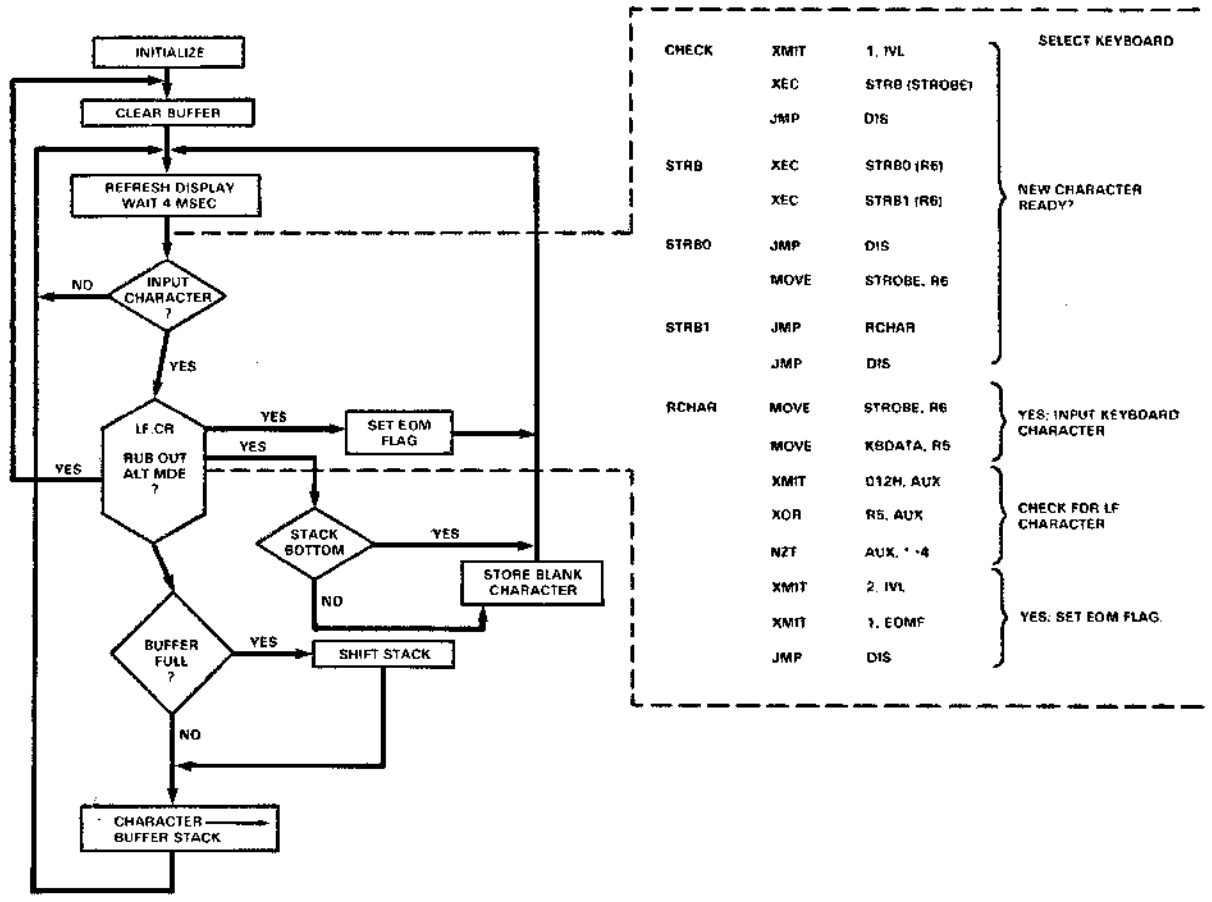


Figure 8

ROM/PROM FOR PROGRAM STORAGE	WORKING STORAGE FOR DATA BUFFERS	IV BYTES FOR INPUT/OUTPUT INTERFACE
Keyboard/driver ..... 140 words	32 bytes per display	41 IV bits for output per display 8 IV bits for input per display Total: 7 IV bytes per display

Table 8 8X300 CONFIGURATION

COMPUTER I/O BUS EMULATOR

DESCRIPTION

The 8X300 system emulates a Microdata 1600 I/O bus. Microdata I/O bus compatible peripherals may then be easily connected to and controlled by a standard 8X300 system. A Microdata I/O bus driver program provides a standard software interface to peripheral devices and requires only 27 words of program storage. Figure 9 illustrates the system.

DESIGN APPROACH

Data bytes are transferred to and from the I/O bus in accordance with Microdata I/O bus specifications. Control signal timing and data transfer sequences are generated by programming. A four step procedure is followed to implement the design:

1. Analyze interface. Analyze Microdata I/O bus relative to: Number of control/data lines, timing and data rates associated with each control/data line, electrical characteristics of each control/data line and determine supplemental circuits needed for electrical compatibility (see Table 9).
2. Perform functional analysis. The functions to be programmed and any which require supplemental logic are determined. In this case, no supplemental logic is required. The programmed functions are:
   a. Transfer bytes in and out
   b. Generate control signal timing and data transfer sequences
3. Define the program to process input and to generate output (see Figure 10).
4. Determine the 8X300 configuration (see Table 10).

COMPUTER I/O BUS EMULATOR

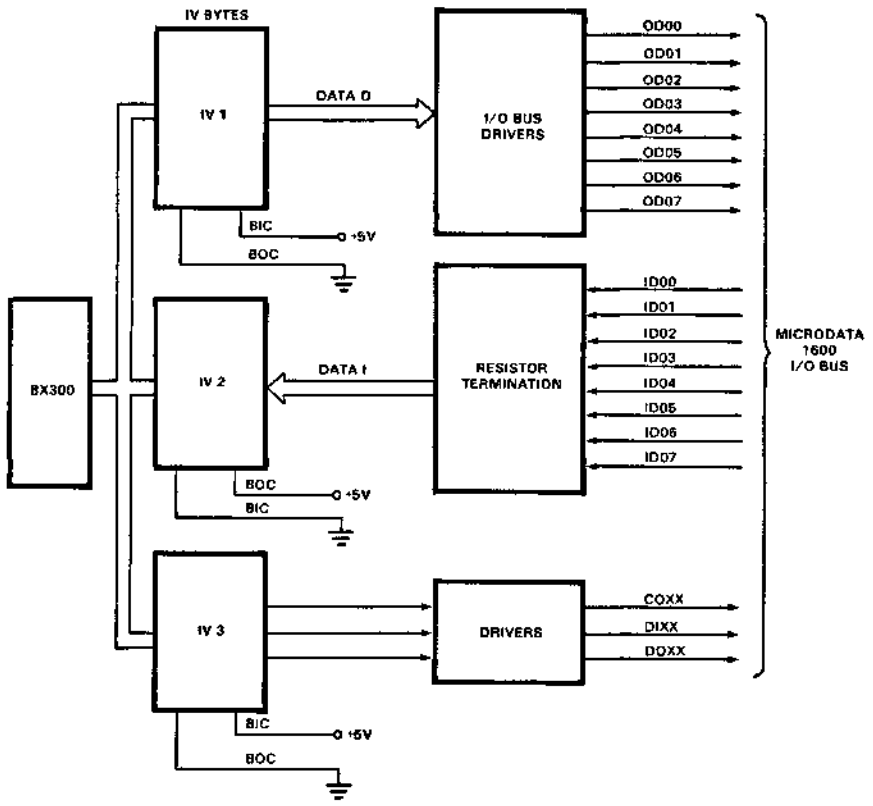


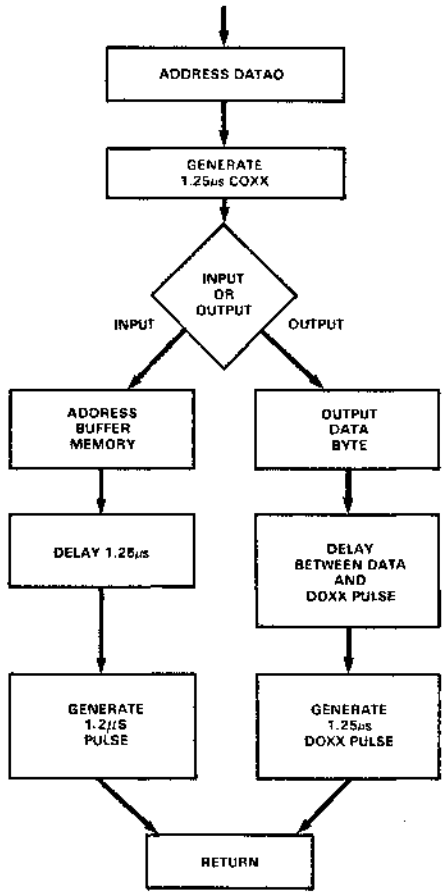
Figure 9

SIGNAL NAME	DATA RATE	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
OD00-07	level		open collector	8	8D	Data/address from computer
ID00-07	level		TTL	8	8R	Data to computer
COXX	4μs	1.25μs	open collector	1	D	Control output timing
DIXX	4μs	1.25μs	open collector	1	D	Data input timing
DOXX	4μs	.75-1.25μs	open collector	1	D	Data output timing

D = Open collector driver
R = Resistors

Table 9 INTERFACE ANALYSIS

FUNCTIONAL ANALYSIS




```
MICRO10   XMIT    1, IVL
          MOVE    R1, DATA0
          XMIT    3,IVL

          XMIT    1, COXX     PULSE ON
          XEC     TAB (R3)
          NOP
          NOP
          XMIT    0, COXX     PULSE OFF

          MOVE    R2, IVR
          NZT     R3, INPUT

OUTPUT    MOVE    WS, DATA0

          NOP
          XMIT    3, IVL

          XMIT    1, DOXX     PULSE ON
          NOP
          NOP
          NOP
          XMIT    0, DOXX     PULSE OFF

TAB       XMIT    1, IVL
          XMIT    2, IVL
```

Figure 10

ROM/PROM FOR PROGRAM STORAGE	WORKING STORAGE FOR DATA BUFFERS	IV BYES FOR INPUT/OUTPUT INTERFACE
I/O Driver ..... 27 words	Depends on peripheral	11 IV bits for output 8 IV bits for input Total: 3 IV bytes per peripheral

Table 10 8X300 CONFIGURATION

INTERFACE TO EXTERNAL READ/WRITE MEMORY

DESCRIPTION

The 8X300 controls the storage, retrieval and processing of large blocks of data. Data is stored in a large capacity (up to 64K bytes) read/write RAM external to the 8X300 system. The memory is assembled from widely available n-channel (n-MOS) static or dynamic RAM circuits. Minimal interface circuitry is required to connect the 8X300 Interface Vector bytes to the address, data and control lines of the external memory. Figure 11 illustrates the system.

DESIGN APPROACH

Data bytes are read from or written into memory through a single IV type. Two additional IV bytes are used as a 16-bit address register to the external memory. 16 bits provide an address range of 65K bytes. The read/write control signals to the memory require two IV bits. Instruction sequences are used for memory read and memory write operations to implement 1 to 2 microsecond memory access times. A four step procedure is followed to implement the design:

1. Analyze interface. Analyze n-MOS RAM circuits relative to. Number of control/data lines, timing and data rates associated with each control/data line, electrical characteristics of each control/data line and determine any supplemental circuits needed for electrical compatibility (see Table 11).
2. Perform functional analysis. The functions to be programmed and any which require supplemental logic are determined. The programmed functions are:
   a. Store memory address in IV bytes ADRHI, ADRLO.
   b. Set appropriate read/write control bits
   c. Wait for memory operation complete
3. Define the program to process input and to generate output.
   a. GET instruction sequence to read memory location addressed by contents of IV bytes ADRHI, ADRLO (see Figure 12).
   b. PUT instruction sequence to write data into the memory location addressed by the contents of IV bytes ADRHI, ADRLO (see Figure 13).
4. Determine the 8X300 configuration (see Table 12).

INTERFACE TO EXTERNAL READ/WRITE MEMORY

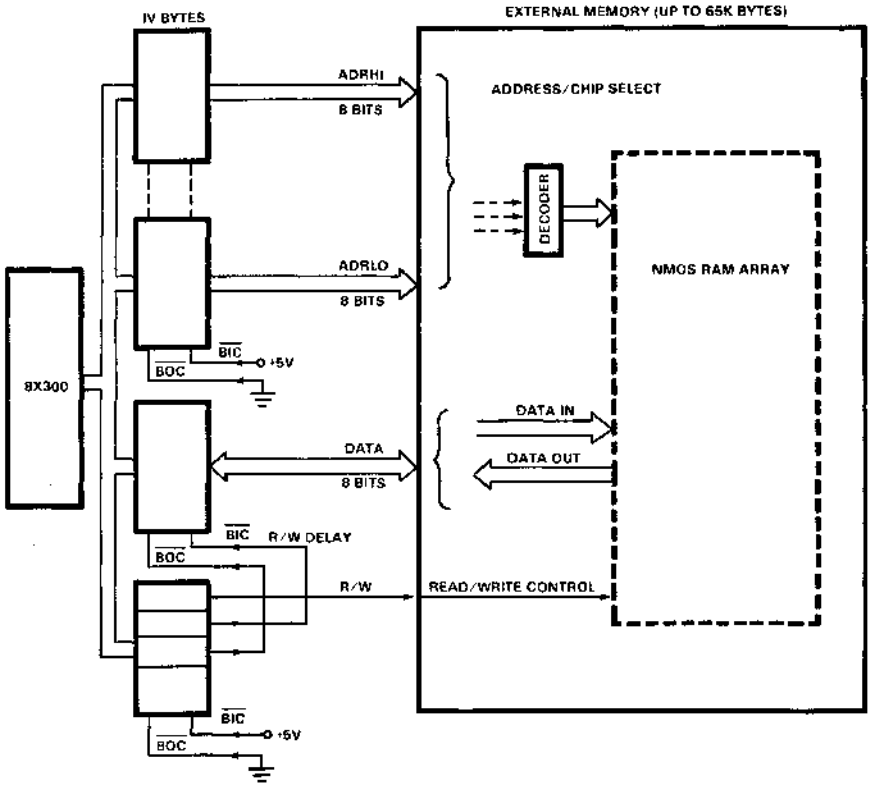


Figure 11

SIGNAL NAME	DATA RATE	SIGNAL DURATION	ELECTRICAL CHARACTERISTICS	# IV BITS	INTERFACE REQUIRED	FUNCTION
ADRHI	Level		TTL	8	* none	Most significant byte. Memory address, and chip select input
ADRLO	Level		TTL	8	* none	Least significant byte memory address
DATA	Level		TTL	8	* none	Memory data
R/W	500ns (min)	>250ns	TTL	1	* none	Memory read/write control
R/W DELAY	500ns (min)	>500ns	TTL	1	* none	Data enable delay during memory write

Table 11 INTERFACE ANALYSIS

GET INSTRUCTION SEQUENCE

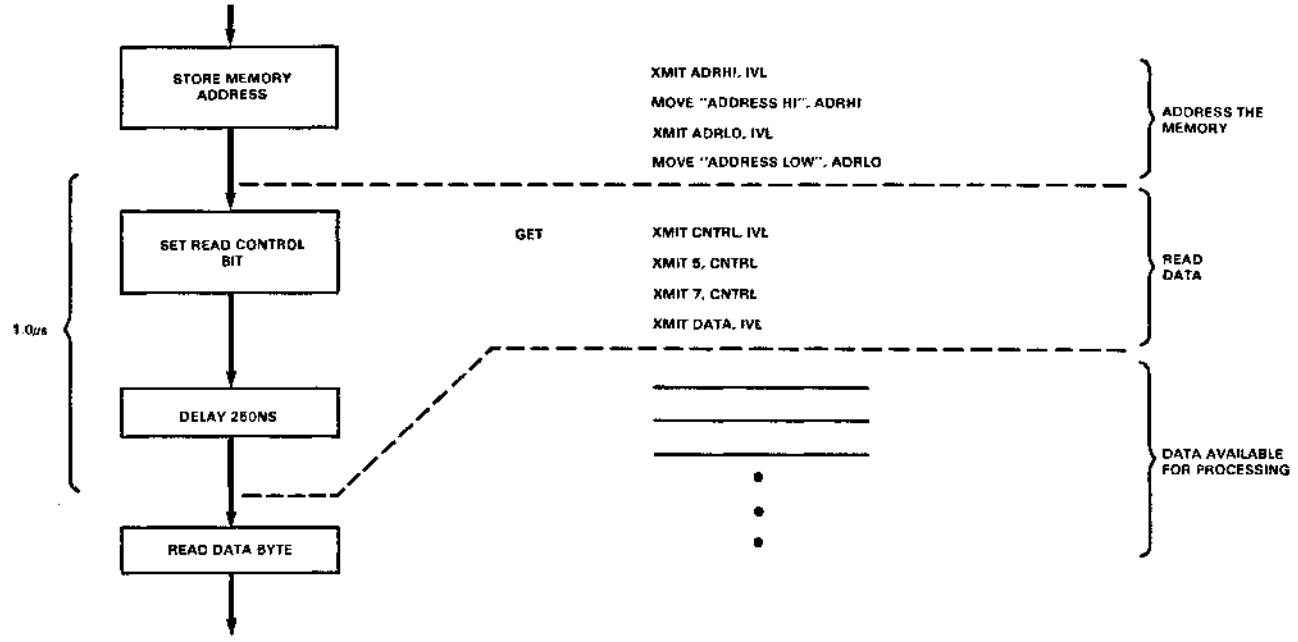
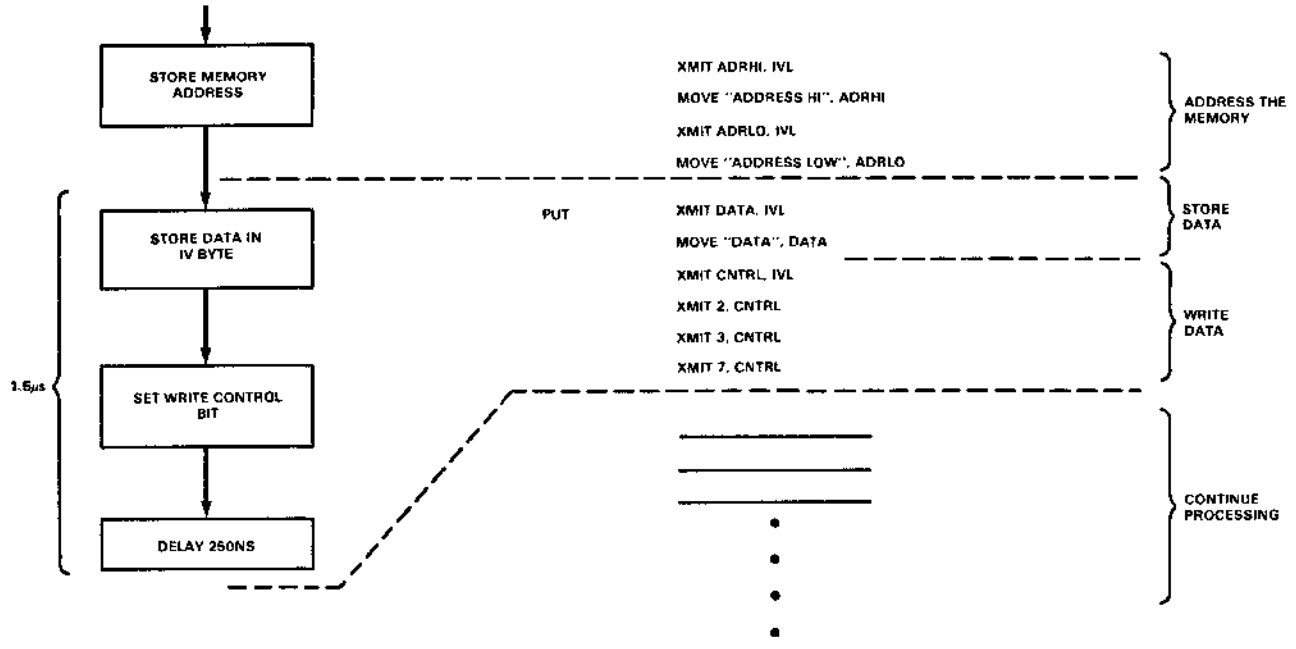


Figure 12

PUT INSTRUCTION SEQUENCE



Throughput:
Single byte transfer time
Memory read: 833k bytes per second
Memory write: 555k bytes per second
For multiple time transfers, throughput is reduced by the time to loop and update addresses and byte counts to approximately:
Memory read: 250k bytes per second
Memory write: 200k bytes per second

Figure 13

ROM/PROM FOR PROGRAM STORAGE	WORKING STORAGE	IV BYTES FOR INPUT/OUTPUT INTERFACE
GET sequence ..... 4 words PUT sequence ..... 6 words	None	18 IV bits for output 8 IV bits for input and output Total: 4 IV bytes

Table 12 8X300 CONFIGURATION

256 WAY BRANCH

DESCRIPTION

Many data communication applications require conversion of one code structure to another. The 8X300's Execute instruction provides a fast and efficient method of performing this conversion.

A single Execute instruction can provide up to a 256 way branch based on a byte stored in a register.

This assumes one of the 256 values does not occur during operation of the Execute table. This is easily prevented by testing for one of the values before entering the table, thereby completing the 256 way branch. The example in Figure 14 details how the test for R1 equal to zero is performed first (NZT). If zero, the appropriate conversion value is loaded into R3 (XMIT). If not zero, then the Execute table determines which of the other 255 combinations is in R1 and loads the appropriate conversion value in R3.

The 256 way branch requires 260 words of program storage and 1.0 microseconds maximum to execute. The Execute table and the Execute instruction must all be located with one 256 byte page where the first instruction address contains zeros in the 8 least significant bits. The other four instructions may be placed anywhere within the 8X300's address space.

8X300 EXECUTE INSTRUCTION

```
CONVERT:    NZT     R1, PAGE        ;BRANCH IF R1 ≠ 0.
            XMIT    VALUE, R3       ;SET R3 BASED ON R1 = 0
            JMP     NEXT
PAGE:       JMP     TABLE

            XMIT    VALUE, R3       ;MUST BE AT EVEN PAGE ADDRESS
              |
              |   255 XMIT INSTRUCTIONS SET
              |   R3 TO A VALUE BASED ON
              |   CODE COMBINATION IN R1.
              |
TABLE:      XEC     TABLE (R1)      ;TABLE = 377.
NEXT:       •
            •
            •
```

PROGRAM STORAGE—260 WORDS
MAXIMUM EXECUTABLE INSTRUCTIONS—4; 1.0μS

NOTE

Value is the appropriate conversion code for each code combination in R1.

Figure 14

FAST IV SELECT

DESCRIPTION

The fast IV select is implemented by adding bits to the instruction word, in increments of 4 or 8 bits. This technique allows IV bytes and working storage to be selected within the same instruction where it is used. This can save important processor time by saving one instruction cycle for each select instruction. It eliminates the need for the IV select instruction. It trades fewer instruction cycle times for hardware. It also trades 16-bit select instructions for 4 to 8-bit select fields, thus saving 8 to 12 bits of program storage for every select instruction saved. To some extent, this reduces the cost impact of a larger instruction word. The technique can be used on both IV and buffer storage (including working storage). When used on IV, a decoder is used following an address hold latch to select one IV per address combination. Buffer storage does not require the decoder, instead it utilizes the address directly.

The fast select IV can be used on the same system with normal select IV since all the fast select IV contains the same address. The Master Enable (ME) input of each fast select IV is enabled by the AND of Bank Select (LB, RB) and the single line decode.

Due to memory access delays, the clock used to latch the fast select address is delayed with a couple of inverter delays to assure address validity. On large systems, there are extra delays which may require the address to be programmed in the instruction prior to its usage. Then a double set of address hold latches are used so the address will appear sufficiently early.

FAST IV SELECT CONFIGURATION FOR SMALL SYSTEMS

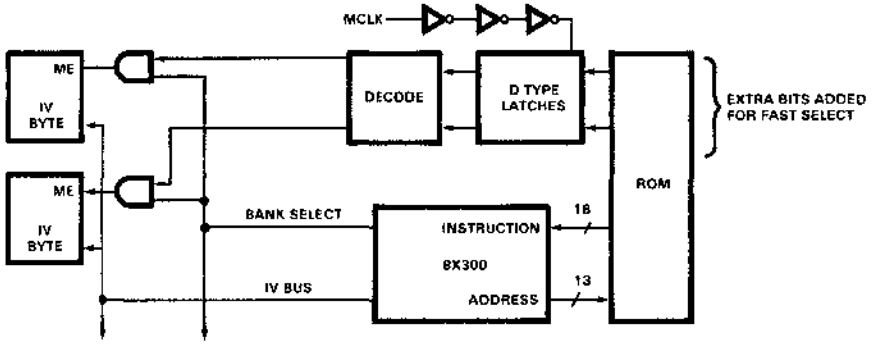


Figure 15

CONFIGURATION FOR LARGE SYSTEMS

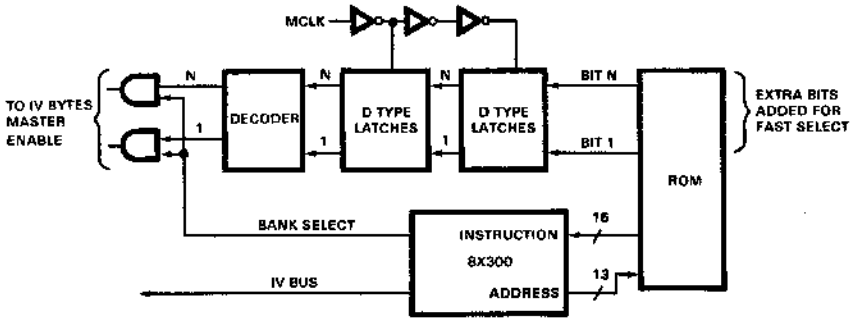


Figure 16

SUMMARY

A study was made as to how a CRT controller can be realized using the Signetics 8X300 Microcontroller. The target specification was based on the specification of the CRT used with the Signetics TWIN system. The same keyboard is used for manual input. To interface with RS232, the PCI 2651 (Programmable Communication Interface Circuit) is used. The screen memory is built up with 21L02-3 and stores 25 rows of 80 characters.

The output of the system can be fed into a standard TV set. The total package count of the digital section of the system is 52 packages (77 DIL equivalents). This is a reduction by half of the package count in the original CRT. The program to run the 8X300 system can be stored in 2 Signetics 82S115 PROMs, as the number of instructions can be kept within 512 words.

Addition of 2 PROMs (82S115) for program store can boost system performance to an intelligent terminal. To have guaranteed operation at worst case timing in RAM access from the 8X300 as well as in access for screen refresh, the memory devices must have an access time of 400ns or less.

For display of 80 characters in the usable part of a line time of a standard TV set (45μs), the access (or cycle) time of the character generator should be 535ns worst case. In this system, the 82S115 PROM is implemented.

INTRODUCTION

This application note describes the design of a system with the Signetics 8X300 Microcontroller for displaying 25 rows of 80 characters on a standard TV set. The characters can be entered from a keyboard or an asynchronous data communication line, interfaced with the Signetics PCI 2651.

The system can operate as a computer terminal in full and half duplex mode at 8 data baud rates (max. 9600 baud). It performs all the functions of the CRT terminal used with the Signetics TWIN, such as the usual cursor actions, clear one line to end of screen or the complete screen. With the cursor at the bottom, a line feed or row overflow will cause a scroll up of the pictures. The capabilities of the system easily can be expanded by adding program ROM.

The hardware is described in detail and software flows are given and explained. The assembly in MCCAP was made and resulted in 503 program steps.

THE TV SIGNAL

For display of dot matrix characters, the video signal has three levels:

1. The white level—if a spot has to be displayed (100%)
2. The black level—wherever no spot is displayed (30%)
3. The synchronous level—to generate line and frame synchronization (0%)

THE TV DISPLAY LINE

A TV signal is built up by writing a number of lines on a screen. At the end of the line, the signal goes to a synchronization level, for about 4.7μs, thereby resetting the line deflection. Figure 1 illustrates the build up of the TV picture. The display of the line is inhibited when the screen boundary is reached and again enabled when the flyback is completed.

Figure 2 shows the complete video signal that has to be produced. The flyback of the spot occurs during the blank time after the line synchronization. During the whole blank time, the display of the next display line can be prepared.

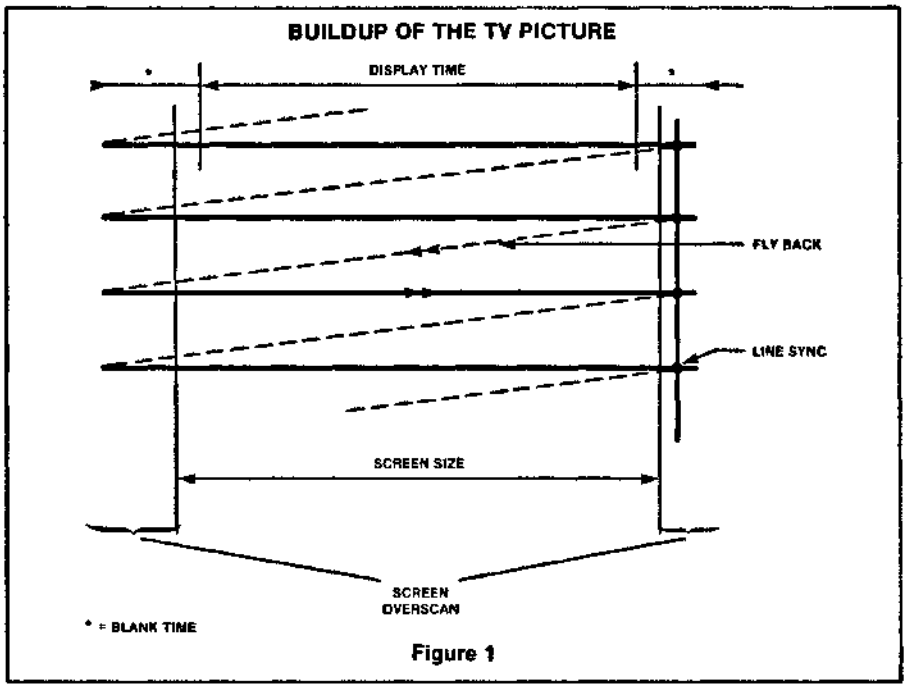


Figure 1

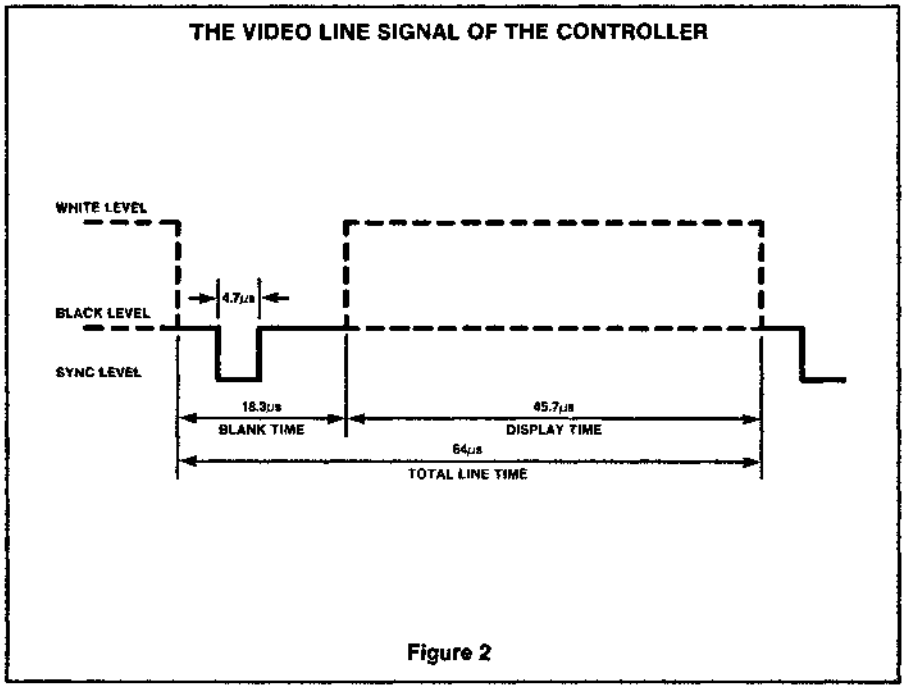


Figure 2

BUILDING UP A CHARACTER ROW

The characters are built up in a 5X7 dot matrix. The character height is 7 lines, the vertical character separation is 3 lines, so that the complete row consists of 10 lines. All the characters of the row are scanned 7 times in each frame to obtain complete symbols on the screen. Figure 3 illustrates the character generation.

FRAME SYNCHRONIZATION

The complete frame includes 25 display rows and a number of blank rows. These blank rows are at the top and the bottom of the frame. When the bottom is reached at row 27 (line 274) a frame synchronization is generated by inverting the synchronization signal.

The synchronization level is on the video line for 59.5μs and black level during the rest of the line time (4.5μs). The frame synchronization is active during 4 line times (lines 274-277). After this frame synchronization, the vertical flyback takes place. The display is restarted (319-277) = 42 line times (42X64μs = 2.7ms) after frame synchronization.

VIDEO TO TV INTERFACE

The display and synchronization signals are fed to a Colpitts oscillator as shown in Figure 4. The oscillator output can be directly connected to a standard TV set on the VHF band.

HARDWARE DESCRIPTION

The block diagram of Figure 5 gives the main parts of the system:

1. The processor with the program store.
2. The interface to keyboard and communication line.
3. The screen image memory and memory address logic.
4. The timing counters and the character dot generator.

THE PROCESSOR AND THE PROGRAM STORE

All the IV bytes (8T32) are situated on the left bank of the 8X300. The RAM locations needed in the program are located in the refresh memory, so that the right bank can be used for other inputs. These inputs are three-state drivers, addressed through IV byte 2 and enabled in the input phase if the Right Bank (RB) is specified.

In the 8X300 there are 11 registers assigned as follows:

00	AUX	Accumulator (Auxiliary)
01	SCR	Used as a scratch register
02	CHAR	Used to store characters or as scratch
03	CPOS	Indicates the position of the cursor in a row
04	CROW	Indicates the row number in which the cursor is located
05	REFR	Indicates the memory row that is being refreshed
06	FROW	Counts the number of rows per frame
07	IVL	For selection of IV bytes on the Left Bank (LB)
10	OVF	Contains the carry after each Add operation
11	CMDR	Contains an index for a command table that starts commands or subroutines
17	IVR	For selection of IV bytes/memory on the RB (not used)

Figure 6 also shows how the program store can be extended to double the size. The input phase (INPT) is active when the 8X300 is not outputting data (WC="1") or selecting an IV byte (SC = "1")

KEYBOARD AND COMMUNICATION INTERFACE

Figure 7 shows that the keyboard signals are multiplexed to the data bus by three-state drivers (8T98). Once each frame time (in the 25th row line 9 or frame line 259), the keyboard input is selected through IV byte 2: CNTL by setting a SELK. The 7 bits data and the strobe are read from the RB and stored in CHAR, the character register. Also, repeat/break/online are tested in the following cycles to decide what has to be done with the character.

The 8 "system" inputs are selected when the keyboard or the baud rate inputs are deselected. Besides the 3 keyboard signals, this input includes 3 PCI signals, the Full Duplex/Half Duplex switch and the LENDBN. This LENDBN is polled at the end of the program block to synchronize with the display and to be able to update the row start address in time.

The baud rate switch is selected each frame (row 27, line 9). The corresponding code is set to the PCI mode 1 and 2 register.

THE CHARACTER GENERATION

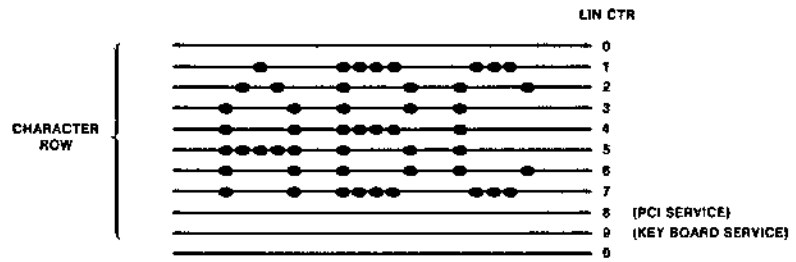


Figure 3

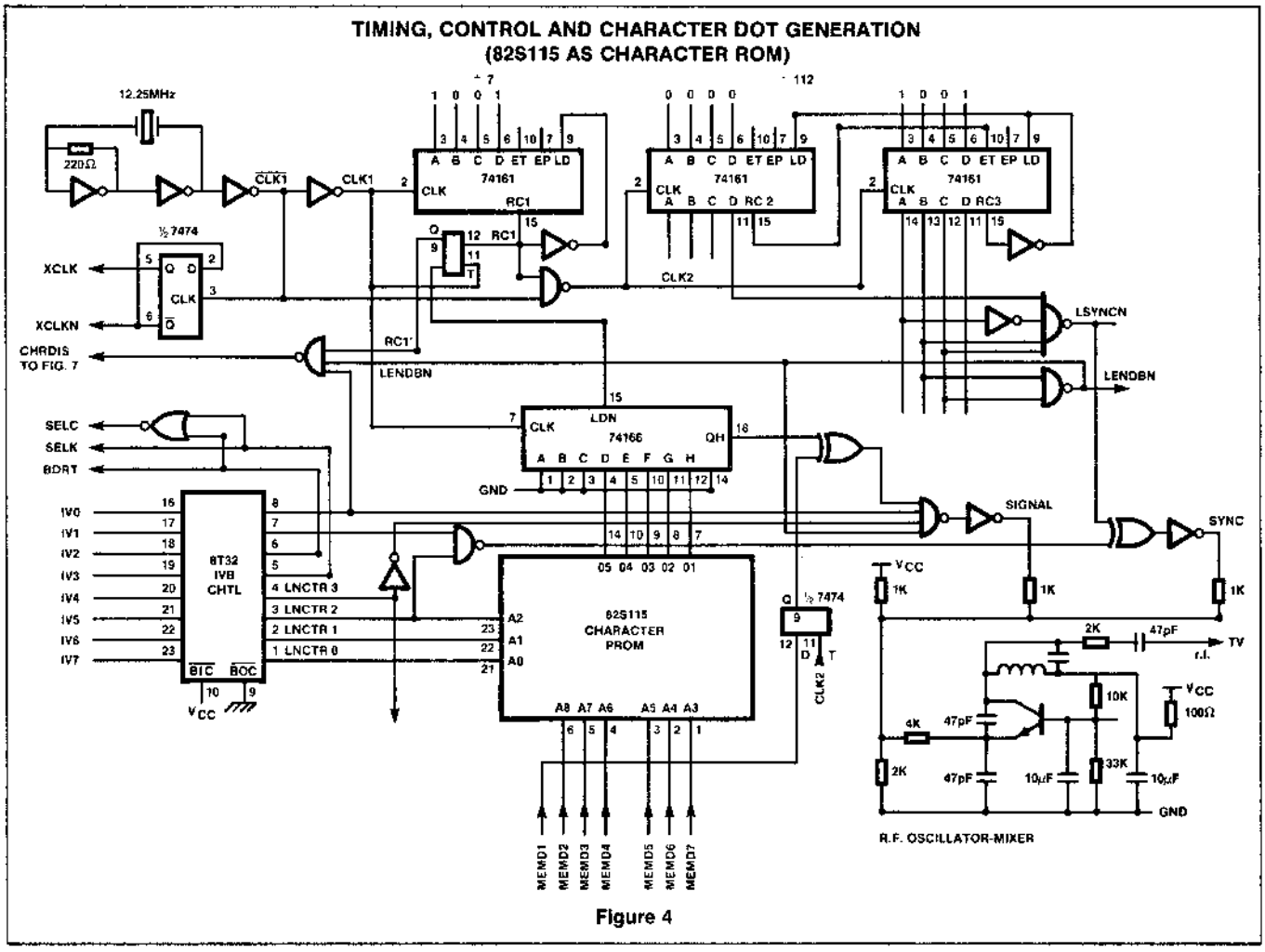

TIMING, CONTROL AND CHARACTER DOT GENERATION
(82S115 AS CHARACTER ROM)
12.25MHz
220Ω
÷ 7
÷ 112
74161
CLK1
RC1
CLK2
RC1'
LENDBN
½ 7474
XCLK
XCLKN
CHRDIS
TO FIG. 7
LSYNCN
74166
LDN
QH
GND
SELC
SELK
BDRT
IV0
IV1
IV2
IV3
IV4
IV5
IV6
IV7
8T32
IVB
CHTL
BIC
BOC
LNCTR 3
LNCTR 2
LNCTR 1
LNCTR 0
82S115
CHARACTER
PROM
SIGNAL
SYNC
1K
2K
47pF
TV
r.f.
10K
100Ω
4K
33K
10μF
GND
R.F. OSCILLATOR-MIXER
MEMD1
MEMD2
MEMD3
MEMD4
MEMD5
MEMD6
MEMD7

Figure 4

BLOCK DIAGRAM OF THE 8X300 CRT CONTROLLER

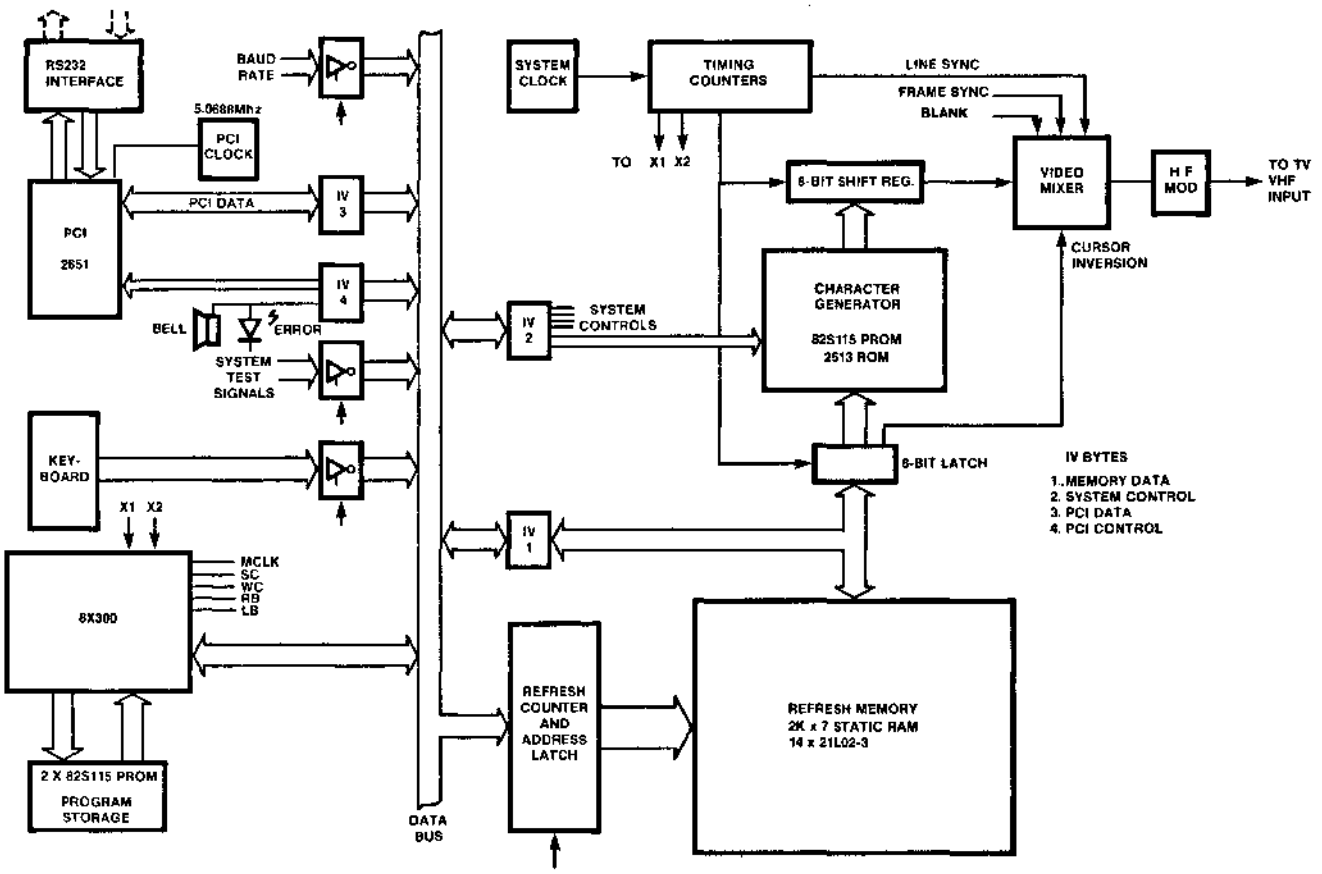


Figure 5

MICROPROCESSOR AND PROGRAM STORE

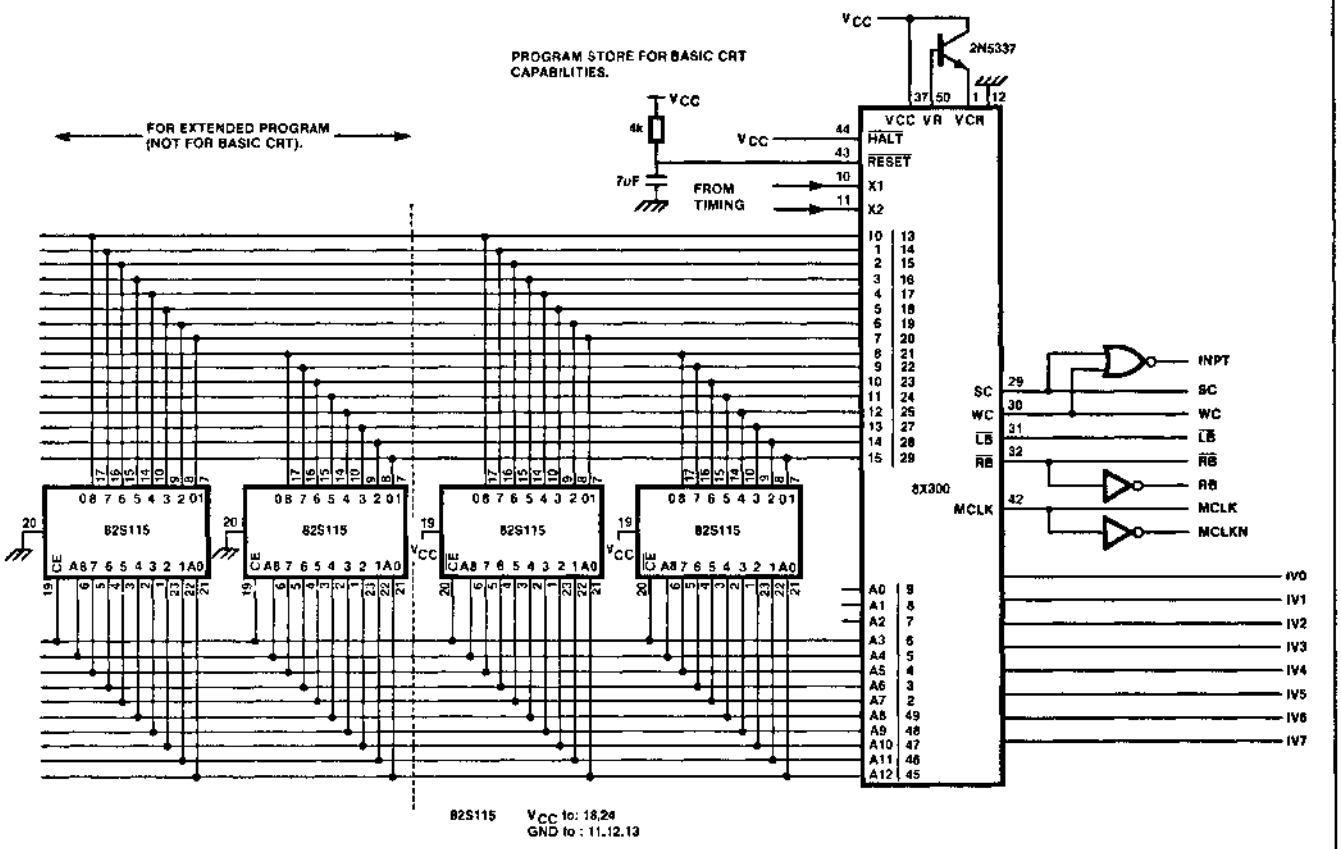


Figure 6

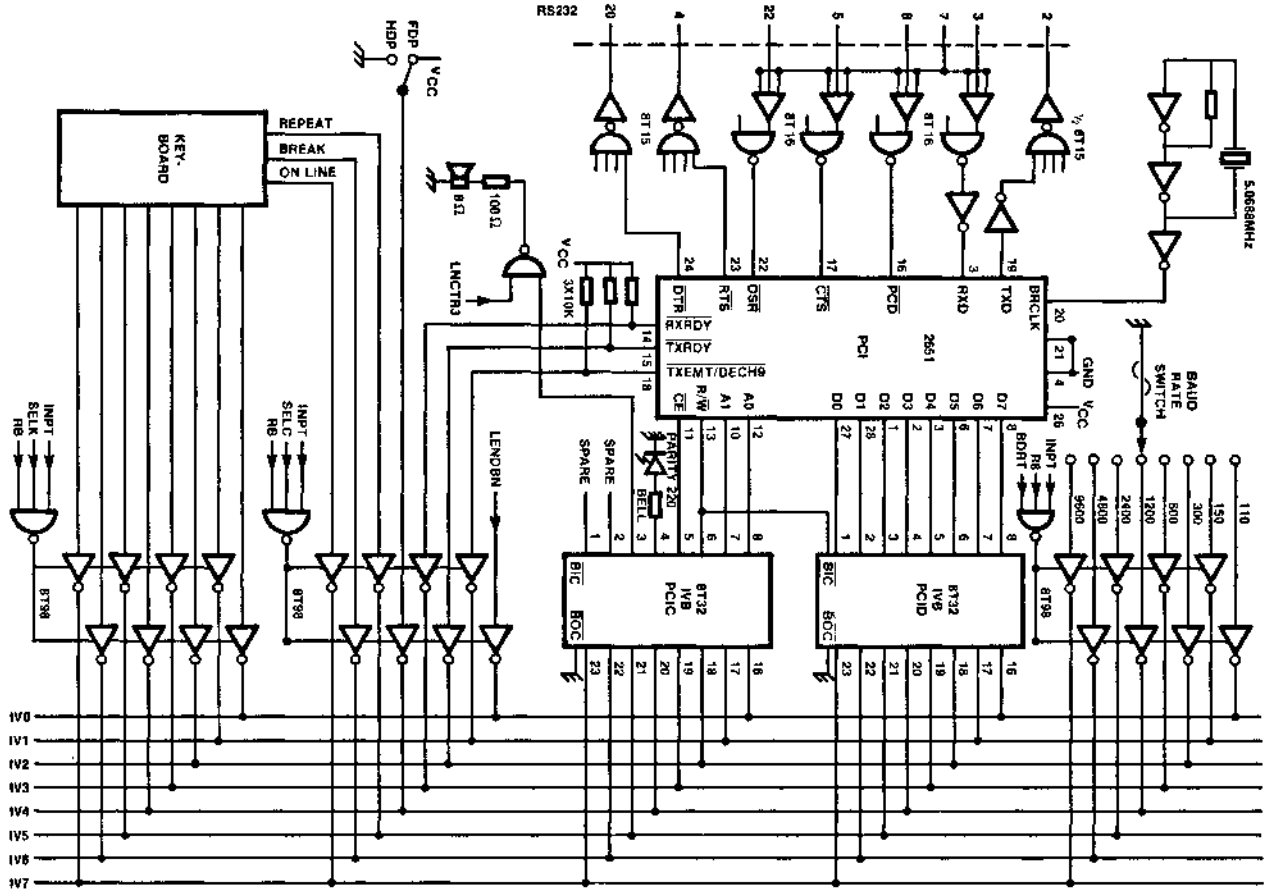

KEYBOARD AND COMMUNICATIONS INTERFACE
RS232
KEY-BOARD
REPEAT
BREAK
ON LINE
HDP
FDP
VCC
100Ω
LNCTRL
3X10K
DTR
RTS
DSR
CTS
RCD
RXD
TXD
BRCLK
RXRDY
TXRDY
TXEMT/DECH9
CE
R/W
A1
A0
D0
D1
D2
D3
D4
D5
D6
D7
PCI
2651
GND
VCC
BAUD RATE SWITCH
5.0688MHz
INPT SELC RB
LENDBN
SPARE
SPARE
PARITY
BELL
BIC
BOC
8T32 IVB PCIC
8T32 IVB PCID
8T98
BDRT INPT RB
9600
4800
2400
1200
600
300
150
110
IV0
IV1
IV2
IV3
IV4
IV5
IV6
IV7

Figure 7

The PCI interface is done with 2 IV bytes. Using IV bytes is necessary since signal conflicts could occur on the data bus: Control data is needed while data is valid on the PCI data inputs when writing into the PCI, control must be stable while data is read from the PCI. These conditions lead to the necessity of latches for both data and control in a 8X300 system. Because data must be routed from and to the PCI, an 8T32 is the only solution.

An IV byte for control gives the possibility of using spare lines for other purposes (in this case, indicators). For PCI to RS232 level interface, the 8T15 and 8T16 level converters are used.

The PCI needs a clock of 5.0688MHz. It would be possible to derive a 12.672MHz dot frequency and this PCI clock from a single 25.344MHz clock, but this higher frequency means that an overtone crystal oscillator must be used and a ÷ 5 counter must be added. Therefore, use of 2 separate crystal clock generators is the most economical solution.

THE SCREEN IMAGE MEMORY

The reason for using the IV byte MEMD as shown in Figure 8 is as follows. Data output from the 8X300 is available in the output phase. If other than bipolar memories are used, the data must be stable in the write cycle for a longer time that half of an 8X300 cycle. For N-MOS memories, an IV byte must be used to latch in the data.

The memory has a size of 2048 words of 7 bits, and thus needs 11 address lines. During refresh of a line, 80 characters of a row are accessed in the visual part of a line time (45.7μs). This means that refresh of lines prescribes a maximum total memory access time of $\frac{45.7\mu s}{80} = 0.571\mu s = 571ns$.

The 11-bit address counting also has to be done within 571ns. Double byte counting in IV bytes would cost at least 6 8X300 cycles per address step, or 6X250ns = 1500ns. Therefore, the address counting is done with an external counter. The 74191 synchronous counter is used. If disabled, it can be used as an address latch.

An access of a memory location must be done by a fixed sequence of operations (see also Figure 9). The 74164 is an 8-bit serial-in parallel-out shift register that is used to control this sequence. Memory access is started by shifting a "1" into the register.

In the 2 cycles that follow, the address is loaded into the counter/latch. During the time that the third positon of the 8-bit shift register is high, the memory is accessed and the memory data byte MEMD must be selected. In the last 2 cycles of the memory access, data can be read and changed.

In assembler (MCCAP) language, this sequence will be:

SEL	MEMA	Select memory sequence
MOVE	ADRLOW,MEMA	Output of low address bits
MOVE	ADRHIG,MEMA	Output of high address bits
SEL	MEMD	Select data IV byte, memory access/time
MOVE	MEMD,REG1	Read data to register REG1
MOVE	REG2,MEMD	Write register REG2 into location

If the last operation is omitted, then the data will not be modified.

The memory access time starts at the moment the address is available at the outputs of the counter/latches. Access must be completed sometime in the input phase of the read cycle. The memory timing shown in Figure 9 shows that the maximum access time of the RAM should not exceed (1.75t_{CYCLE}-163ns), where the t_{ACC} (1.75 X 327 - 163ns) = 409ns.

The 21L02-3 has a worst case access time of 400ns. (The 8T98 inverters for the cursor bit are never accessed in the current setup. Access of the cursor bit leaves exactly 400ns access time for memory and bus delays).

CHARACTER REFRESH TIMING

The character refresh is started after LENDBN is returned to "1" (see Figure 10). The character address is clocked to the second character of the row 1 dot step (CLK 1) after the return of LENDBN, while the dot information of the first character is clocked into the 74166 shift register. The cursor inversion bit is latched separately at the same time.

The timing diagram is shown in Figure 10. Every 7 pulses, the shift register is loaded and the character address is incremented. The cursor indication is made with an exclusive-OR with the cursor bit. The dot signal is mixed with the synchronization signal and modulates the rf oscillator.

LINE SYNCHRONIZATION AND LINE END BLANKING

The line synchronization is generated with 74161 counters from the character frequency. The two's complement of the dividing constant is the present value that is loaded synchronously if the overflow carry is "1."

The line synchronization (LSYNCN) is active "0" for 8 character positions (8 X 571ns = 4.57μs). The line end blanking starts 8 characters before the line synchronization and is active during 32 character positions (32 X 571ns = 18.3μs). The line end blanking inhibits the dot display and the character counting.

FRAME SYNCHRONIZATION

Frame synchronization is controlled via software and the CNTL IV byte. If bit 7 is set, the line synchronization is inverted. The synchronization level will be presented during the line, and the blank level during the line synchronization time. Frame synchronization is "ANDed" with the (LCNTR2) bit in the CNTL byte, and therefore has a length of 4 line times.

SOFTWARE DESCRIPTION

The software to control the hardware and to realize the cursor control and clear actions is stored in the 512X16 ROM. The hardware control is a considerable part of the program. Addition of another 512 words of program store would allow more editing possibilities to be added.

In the program there are 3 types of routines:

1st The main scan loop in which the counters are serviced each line time.
2nd Service routines—periodical scan of the peripheral devices (the keyboard and the PCI).
3rd Command routines, in which the commands are executed.

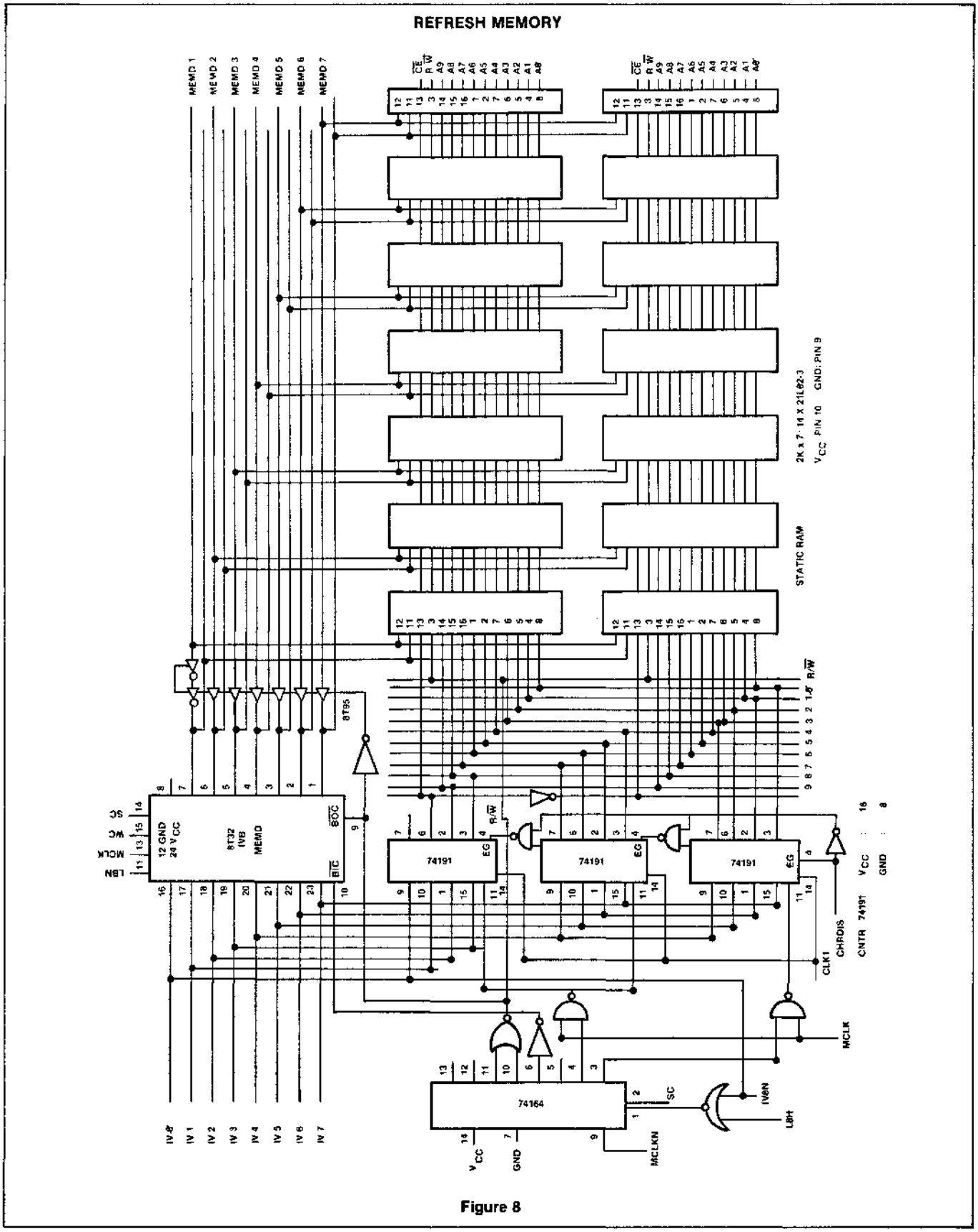
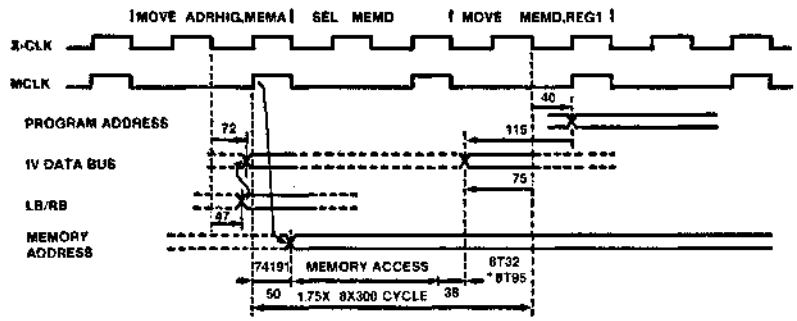

REFRESH MEMORY
MEMD 1
MEMD 2
MEMD 3
MEMD 4
MEMD 5
MEMD 6
MEMD 7
STATIC RAM
2K x 7: 11 X 21L82-3
V_{CC} PIN 10 GND: PIN 9
8T95
8T32
IVB
MEMD
12 GND
24 V_{CC}
74191
74164
CHRDIS
CNTR 74191 V_{CC} : 16
GND : 8
CLK1
MCLK
MCLKN
IVBN
LBH
IV-8
IV 1
IV 2
IV 3
IV 4
IV 5
IV 6
IV 7

Figure 8

MEMORY READ ACCESS CYCLE TIME

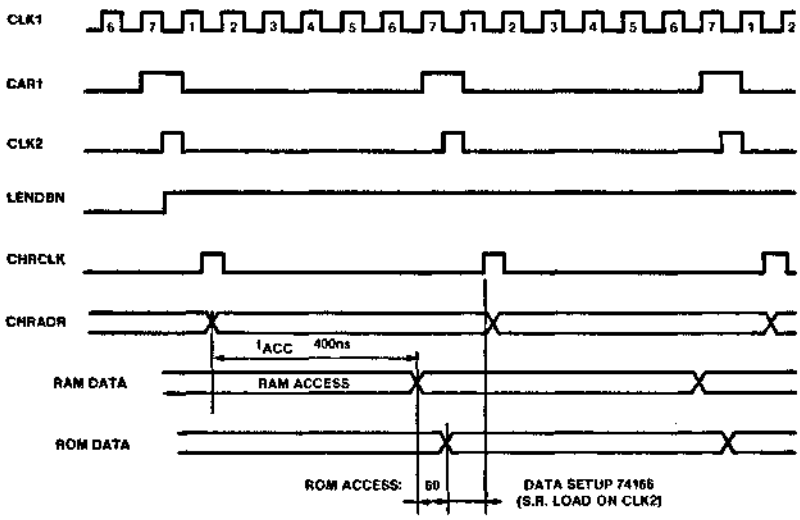


NOTES

1. Memory read access cycle = $t_{ACC} \leq 1.75\ T_{Cycle} - 163ns$
2. The memory address is latched in the 74191 with MCLK and stable after 50ns.
3. The IV Bus must be stable 75ns before the end of the input phase.
4. The IV byte MEMD has a worst case delay of about 25ns, the 8T95 = 13ns.

Figure 9

CHARACTER TIMING IF THE 82S115 ROM IS USED



CHARACTER TIMING

Counter CLK to RAM address	42ns
RAM address to RAM data 21L02-3	400ns
ROM address to ROM data 82S115	60ns
Shift register data setup 74166	20ns
Total	522ns

(available 571ns)

Figure 10

The Main Scan Loop

LINEND

LINEND is the main scan loop (flow 1) which is illustrated in Figure 11. This loop always must be entered before the line end blank signal (LENDBN) is "0." This means that all the routines are executed in blocks that have a limited number of cycles. In the system these blocks have a length less than 184 8X300 cycles.

If the line counter after incrementing is less than 8, it will start the execution of a command or if no command is set, the refresh of a line. If the line counter (LINCTR) is equal to 9, the frame row counter and the refresh row counter (FROW and REFR registers in the 8X300) are serviced.

The FROW is tested in line 9 of each row to be 25 or 27. In row 25, the keyboard is serviced. In line 27, the frame synchronization signal is set and the PCI mode registers are reloaded with values according to the baud select. Frame synchronization reset is automatically done in line 9 of row 28.

Reset is automatically started at power up. Reset is executed from program address 0. It clears the pointers and sets a clear screen command. SETCUR is a return label used in command routines. CMDSR0 starts a command execution from a command table. Before it jumps to the command address, the cursor bit in the refresh memory is reset. This is done because most of the actions include a change of cursor position. COMEND is a label to return from a command if the execution is completed.

PERIPHERAL SERVICE ROUTINES

KEYBD0 is the key board service (flow 2) which is illustrated in Figure 12. The keyboard is scanned each frame time (frame line 259). If a command other than Break is set, then the keyboard is ignored in that frame time.

If the command is a Break, then the break delay is serviced. The break will be sent out during ±500ms (12 frame times). The break command is set together with the break delay after the command test. The break command is reset if the break key is released. In this way, only 1 break pulse can be transmitted per key strobe.

The key strobe for character input is also inhibited following keyboard services if in the former scan the strobe was set. This is done by using a memory location that is set or reset depending on the keyboard strobe level. If the keystrobe memory was "1" in the former scan, the Repeat key is tested and if it is pressed, the keyboard character will be accepted each fourth scan.

FLOW 1—MAIN SCAN LOOP

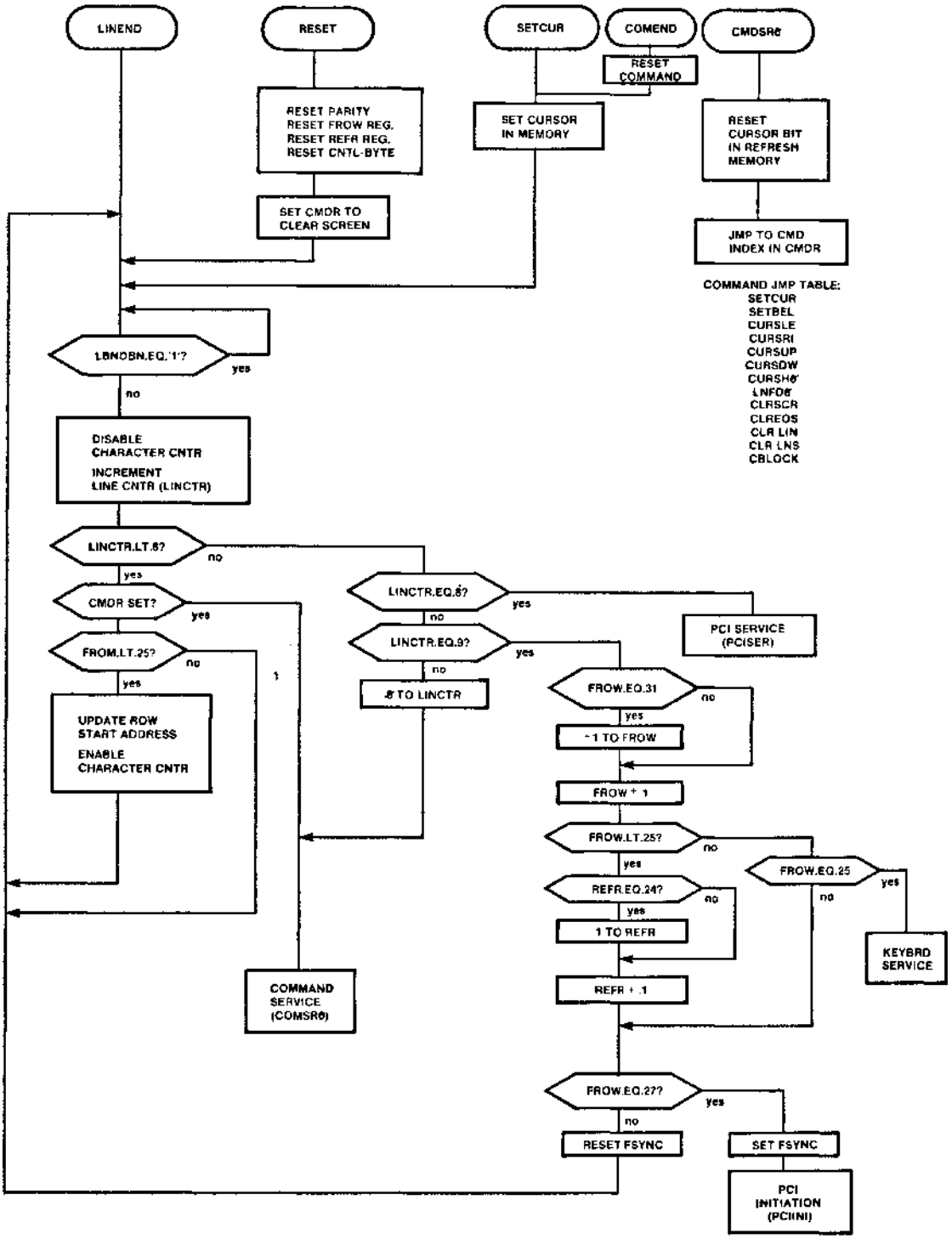


Figure 11

FLOW 2—KEYBOARD SERVICE

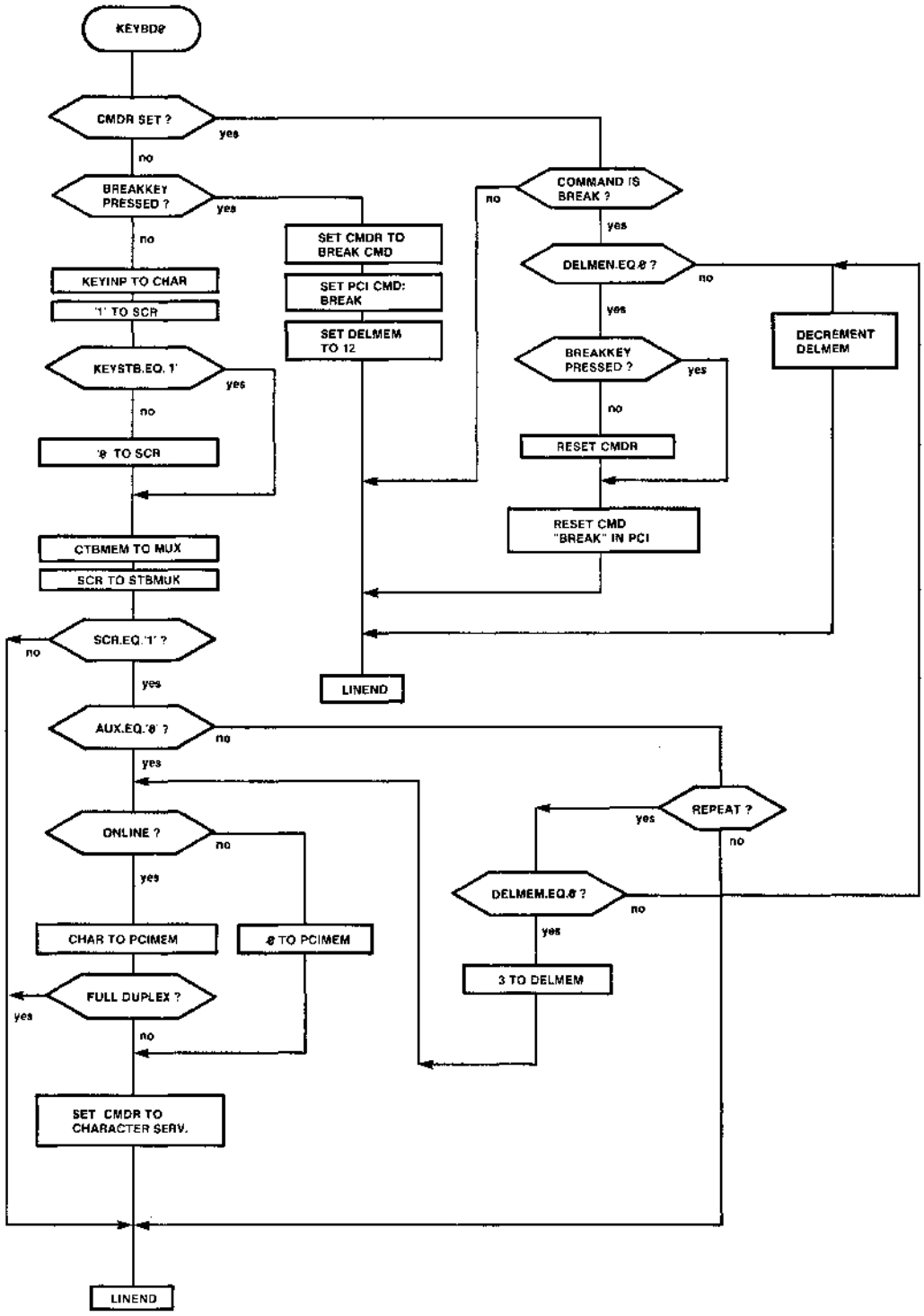


Figure 12

At the end of the keyboard service, the online and half duplex/full duplex switches are tested. If the system is not online, then the PCI data memory PCIMEN (character to be transmitted by the PCI) is reset and the command for the character service routine is set.

If the system is online, then the character to be transmitted from the keyboard is loaded in the PCIMEN location and the FDP/HDP switch is tested. When online, the character service command will be set only in case of half duplex. In full duplex, characters are serviced only if received via the PCI from the data communications line.

PCISER

The PCI service (flow 7) is shown in Figure 13. It starts with a test on the command register. The CMDR will be set only if one of the longer command routines has not been completed.

This type of routine starts with a reset of the DIR (data terminal ready signal). The system that provides signals will either test the DIR signal or must wait automatically after sending these commands. Characters received while a command is still set will result in an overrun error.

If the transmitter of the PCI is ready, the new character to be transmitted is loaded via SCR (an 8X300 register). If the receiver is ready, the data is read into the CHAR register of the 8X300. The status of the PCI is tested for transmission errors. The PCI commands transmitter enable and receiver enable are set if their respective ready signals where active and a non-zero character is transmitted and/or received.

PCIINI

PCI initialization (flow 7) is done every 27th row at line 9. The mode one register setting is independent of the switch setting. For mode two register setting, the baud rate switch is tested and the accessory code is merged with the fixed part of the code. Also in this routine, the bell delay is serviced and the bell signal is controlled.

FLOW 7—PCI SERVICE AND INITIATION + BELL SERVICE

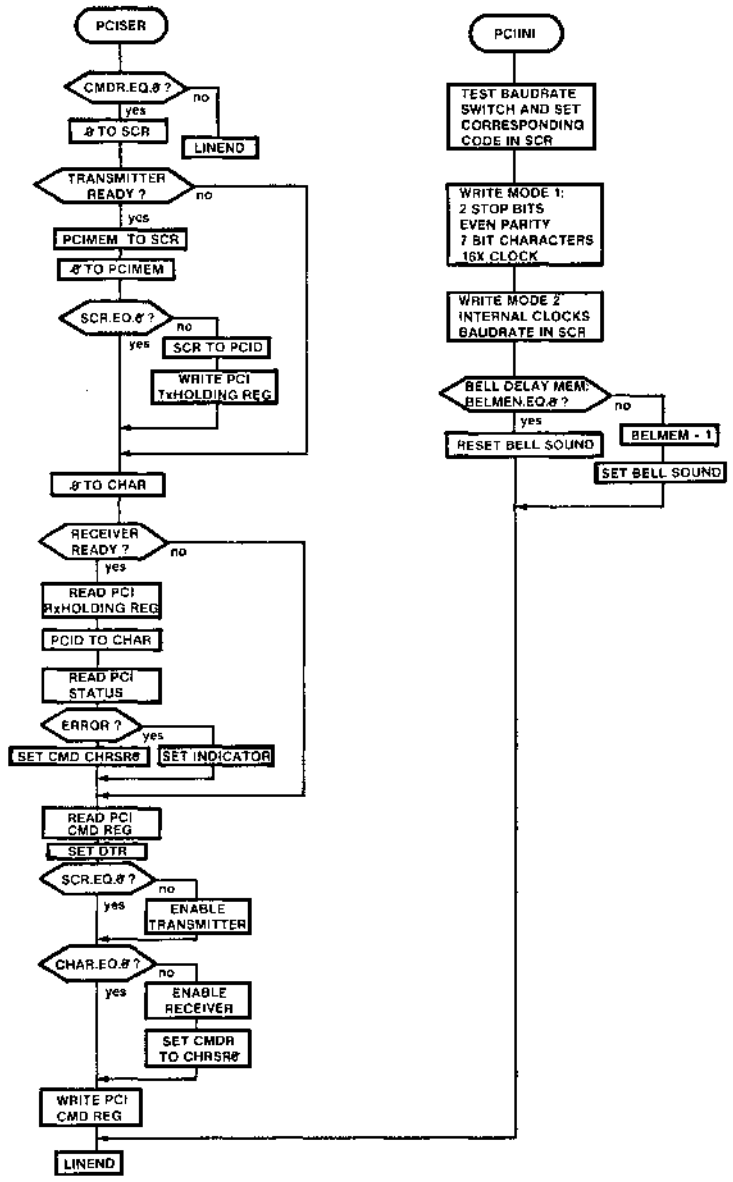


Figure 13

COMMAND ROUTINES

Command routines are started from the main scan loop with an indexed jump. The command register CMDR is the index of the command that is set. The commands start with a reset of the cursor bit and ends with reset of the command register if completed (COMEND) and a restore of the cursor bit (SETCUR).

CHRSR0

CHRSR0 is the character service routine (flow 3) shown in Figure 14. The command to run this routine is set by the keyboard input or the PCI input routines. The character to be serviced is located in the CHAR register. The routine compares CHAR with the escape code.

If CHAR is an ESC, then it sets the ESC memory location and jumps out of routine. If no ESC was detected, the ESC memory is reset. The character is compared with an escape command table if the escape memory was set in the previous chapter service. If the character is not found in the ESC character table after an escape, it will be ignored. When a control character to be serviced and also the previous character are both not the escape code, then the CHAR contents is compared with the control character table. When a control character is found the accessory command is set. If not found, the character is a normal character to be displayed and it will be loaded in the screen memory on the cursor position.

The routine then finishes with an increment of the cursor position. Possible row end actions, like conditions set of bell delay, carriage return and line feed actions will also be done.

FLOW 3—CHARACTER SERVICE

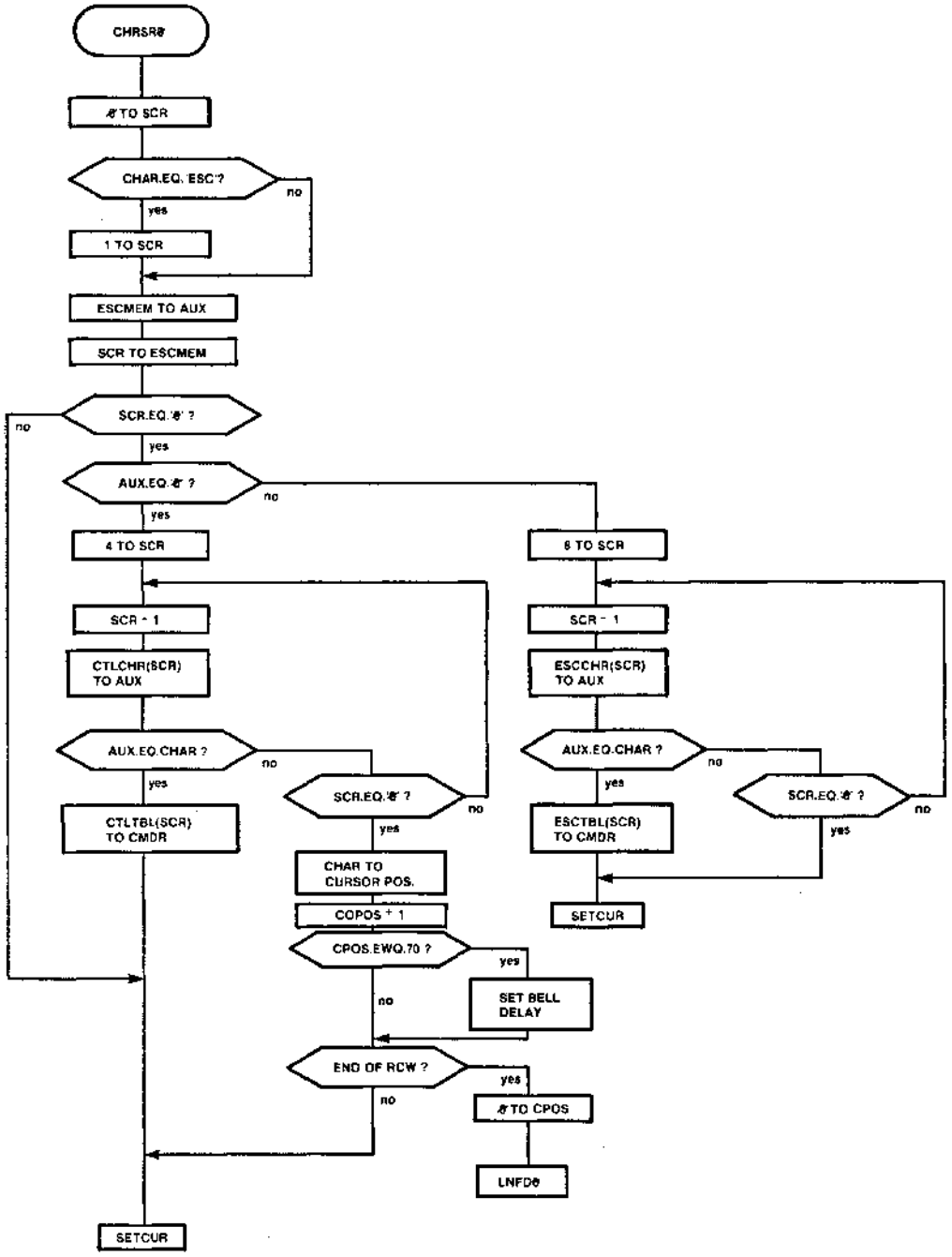


Figure 14

LNFD0
LNFD0 is the line feed routine (flow 4) shown in Figure 15. A line feed can be executed after a control A or at a row overflow from the character service routine. A line feed starts with an increment of the cursor row register CROW. Then a scroll up detection is made as follows.

If the frame row counter FROW is in the blanked rows (FROW not less than 25), then the refresh row counter REFR points to the bottom row of the screen. Scroll must be made if the cursor row is equal to the bottom display row, in this case, the REFR.

If FROW is less than 25, then the bottom display row has to be calculated. The bottom of the screen is reached if FROW is 24. Therefore, the number of rows increments needed to reach this row for FROW and REFR is (24 - FROW). At the bottom row the REFR would be REFR + (24 - FROW). If this value is equal to the cursor row (CROW), then a scroll up must be made in the line feed routine. Scroll is REFR + (24 - FROW) = CROW, which is the same as REFR + 24 = CROW + FROW. The scroll is simply made by an extra increment of the refresh counter (REFR + 1).

SETBEL
SETBEL (flow 4) is a command routine that sets the bell delay memory also shown in Figure 15. Further bell service is done in the PCI initiation routine.

ADDITIONAL CURSOR ACTIONS
The following cursor actions (flow 5) are illustrated in Figure 16 and do not need further explanation.

CURSRI	Cursor right routine
CURSDW	Cursor down routine
CURSLE	Cursor left routine
CURSUP	Cursor up routine
CRET0	Carriage return or cursor return routine
CUSH0	Cursor home routine

FLOW 4—LINE FEED ROUTINE AND BELL DELAY SETTING

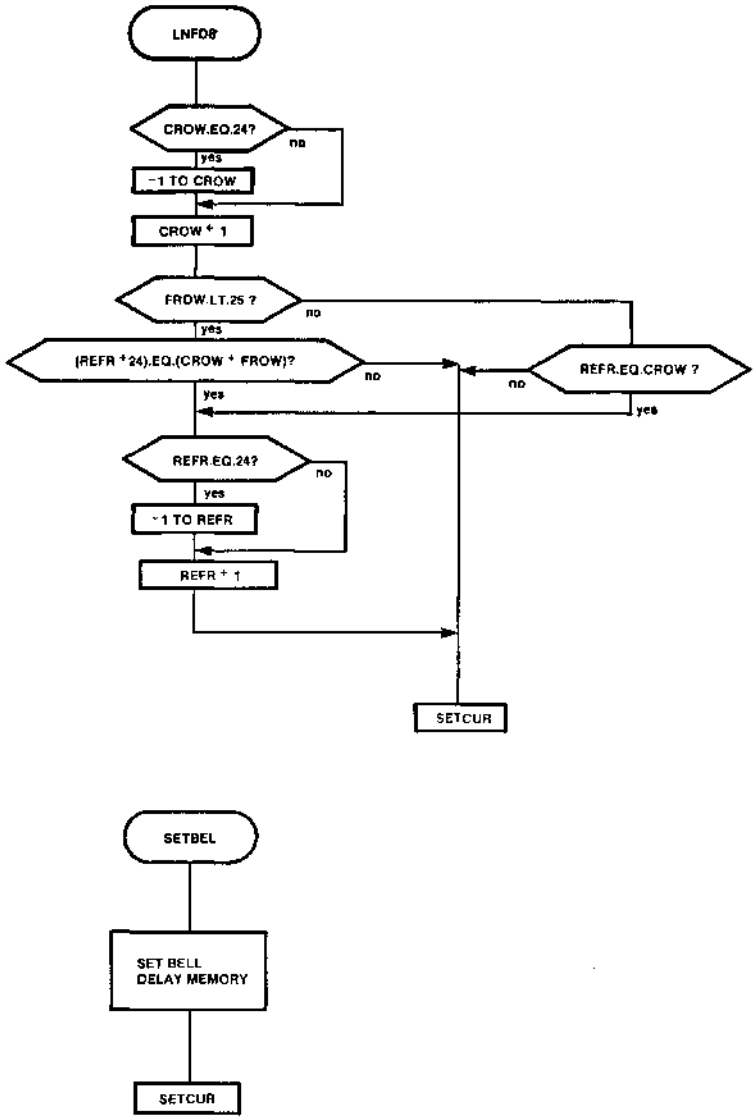


Figure 15

CLEAR ACTIONS

The system performs 3 clear actions (flow 6) shown in Figure 17:

1. Clear line—the row in which the cursor is located is cleared.
2. Clear to EOS—all rows from cursor row to the bottom row are cleared.
3. Clear screen—all rows are cleared, cursor is set to the home position.

The clear commands must be executed in blocks.

CLRSCR
The clear screen starts with setting the cursor home and then continues in the clear to end of screen routine.

CLREOS
The clear to end of screen starts with a load of the bottom row number into the register that points in the clear routines to the line to be cleared (CHAR). This line to be cleared register is compared after the clear line action with the cursor row position.

If the cursor row has been cleared, the command is completed. If the cursor row is not reached yet, CHAR is decremented and the next row will be cleared. In this way, the screen is always cleared from the CHAR row upwards until the cursor row is reached.

CLRLIN
The clear line command sets the line to be cleared register equal to the cursor row. The further clear action can be used by all 3 clear commands, using the CHAR.EQ.CROW compare to see whether more lines have to be cleared. The CPOS register is stored temporarily so that is can be used in the clear routine.

Before ending the initiation of the clear, the DTR signal of the PCI is reset since no data can be received during clear (DIR will be set in the PCI service). The next action of the clear routine will be to clear a block of characters in the following line time.

CBLOCK
Clear block clears a block of 16 characters. The block is determined by CHAR for the row and SCR for the position. The command will set the CLRLNS command if a complete row is cleared.

FLOW 5—CURSER ACTIONS

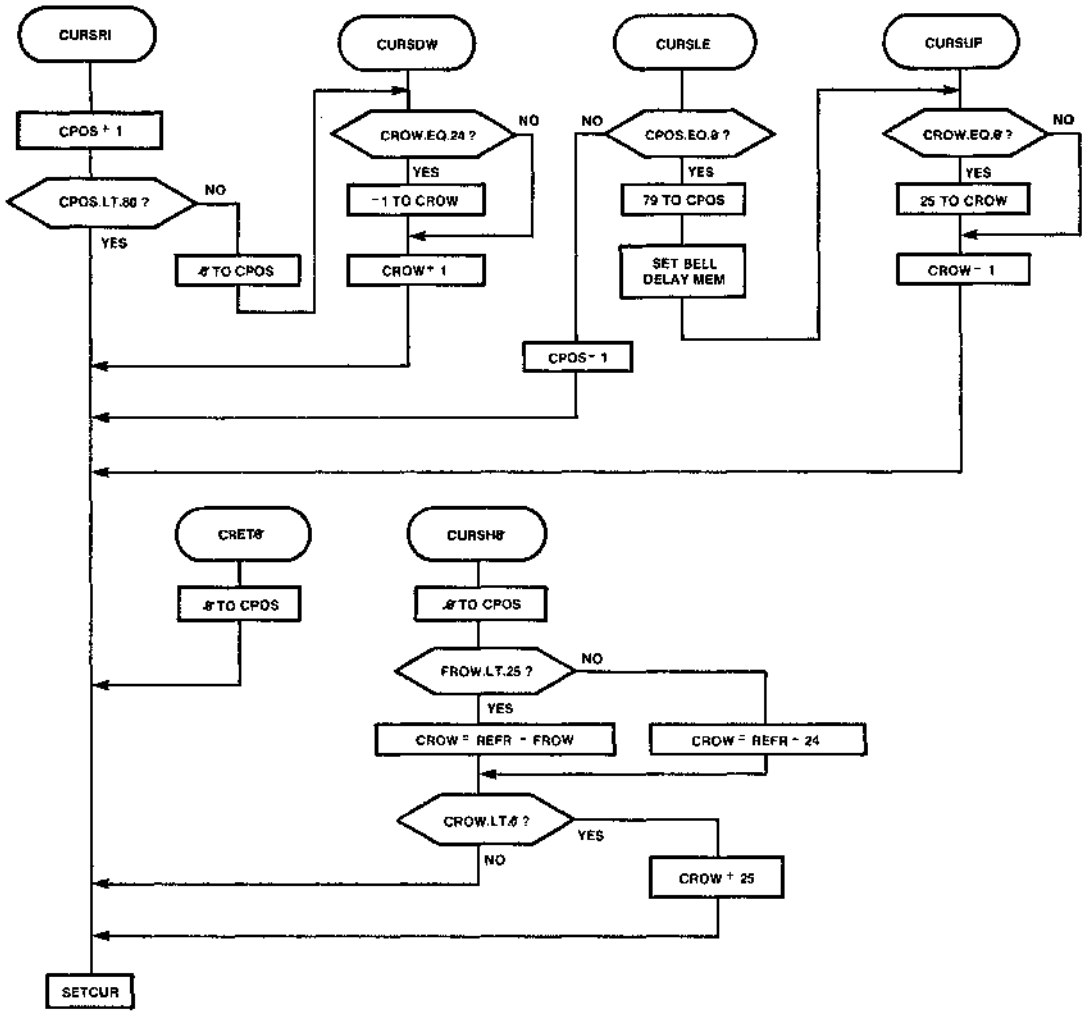


Figure 16

CLRLNS
The clear line command sets the line to be last row is cleared. If so, CPOS will be restored from the memory and clear the command register. If there are more lines to be cleared, CHAR is decremented and the clear of the next row is initiated.

CONCLUSIONS OF THE STUDY

The target specification to make a CRT controller that performs the TWIN CRT functions and is built up with the 8X300 microcontroller is feasible. The total package count of the system is 52. Converted to 16-pin DIL package equivalentes, the board space will be about 77 DIL package places. The package count of the present TWIN CRT is 112 packages. (The used board space could carry 130 DIL packages).

The program to run the 8X300 system can be stored in two 82S115 PROMs, as the number of instructions can be kept within 512 words. To have guaranteed operation at worst case timing in RAM access from the 8X300 as well as access for screen refresh, the memory devices must have an access time of 400ns or less. In this system, the 21L02-3 is used.

For display of 80 characters in the usable part of a line time of a standard TV set (45μs), the access (or cycle) time of the character generator should be 535ns worst case. In this system, the 82S115 PROM is implemented.

FLOW 6—CLEAR ACTIONS

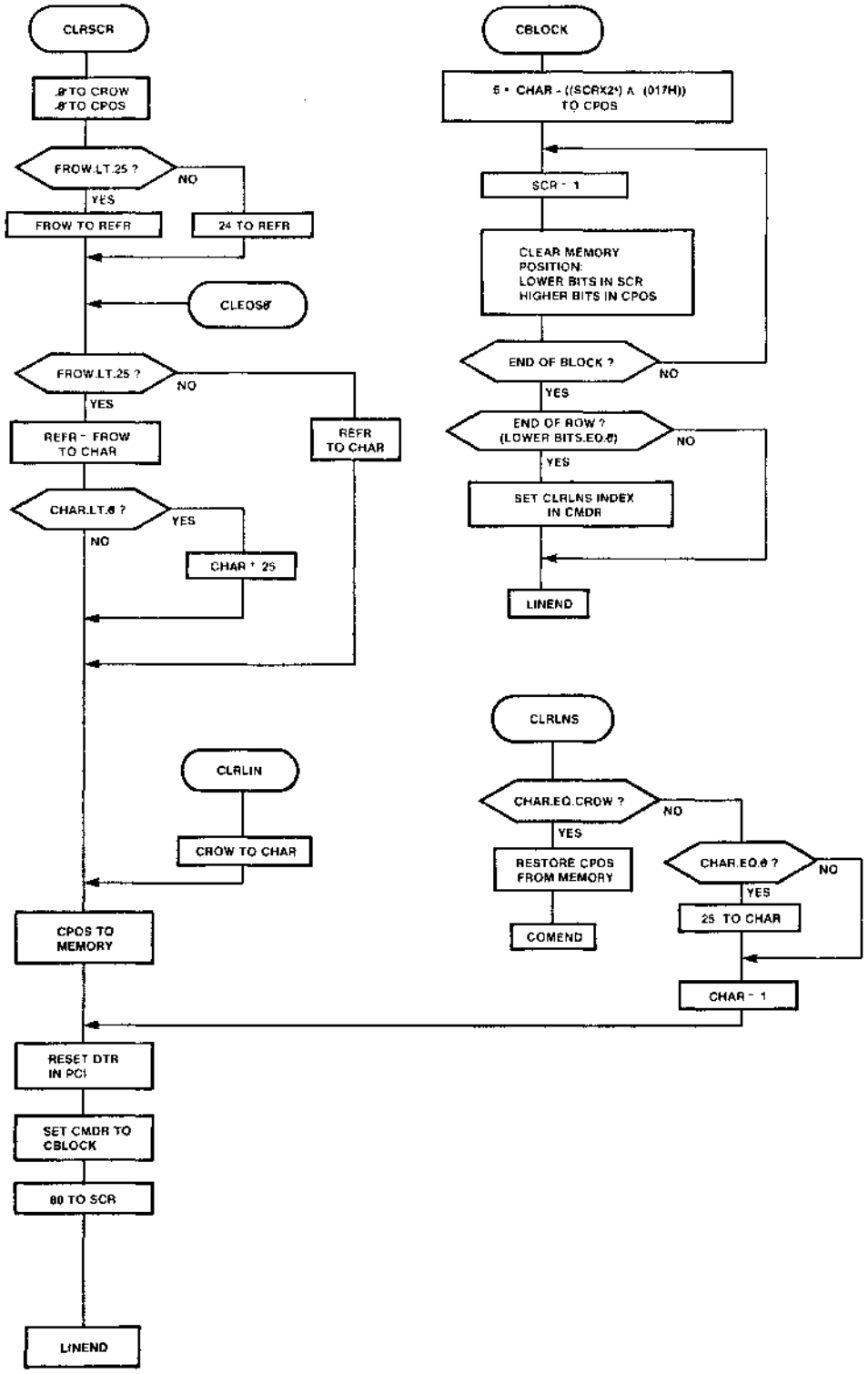


Figure 17

APPENDIX 1
8X300 CRT Controller Package Count

TYPE	FUNCTION	NO. OF PINS	NO. OF PACKAGES	DIL EQUIVALENT*	
8X300	Microcontroller	50	1	1X6	6
82S115	PROM	28	3	3X3	9
8T32	IV Byte	24	4	4X3	12
2651	PCI	28	1	1X3	3
7400	NAND 2	14	2		2
7402	NOR 2	14	1		1
7404	INV	14	3		3
7405	INV (o/c)	14	1		1
7410	NAND 3	14	1		1
7420	NAND 4	14	2		2
7474	IKFF	14	1		1
7486	Ex-OR	14	1		1
74161	8-Bit SR	14	1		1
74164	8-Bit SR	16	1		1
74191	Binary Counter	16	3		3
8T15	Driver	14	2		2
8T16	Receiver	14	2		2
8T98	3-State Inverter	16	4		4
8T97	3-State Buffer	16	1		1
21L02-3	RAM 1K1	16	14	14	14
		Package Count	52	Board Space	77

*The estimation of the DIL equivalents for the LSIs is rather conservative. The bus structure of the system may result in a more economical use of board space.

APPENDIX II
Alternative Character ROM 2513

The 82S115 is a bipolar fusible link PROM with a fast access of 60ns. Use of a much cheaper character ROM leads to some timing problems. Refer to Figures 18-21.

1. The RAM and ROM access times have to be split in to character phases: One phase for character RAM access, one phase for dot ROM access.
2. If no special arrangements are made, the character speed or the number of characters per line will be limited. Due to flyback and screen overscan, the useful time of the line is about 45μs. With an ROM access time of 600ns + latch + SR setup $\frac{45{,}000\text{ns}}{637\text{ns}} = 70$ characters.

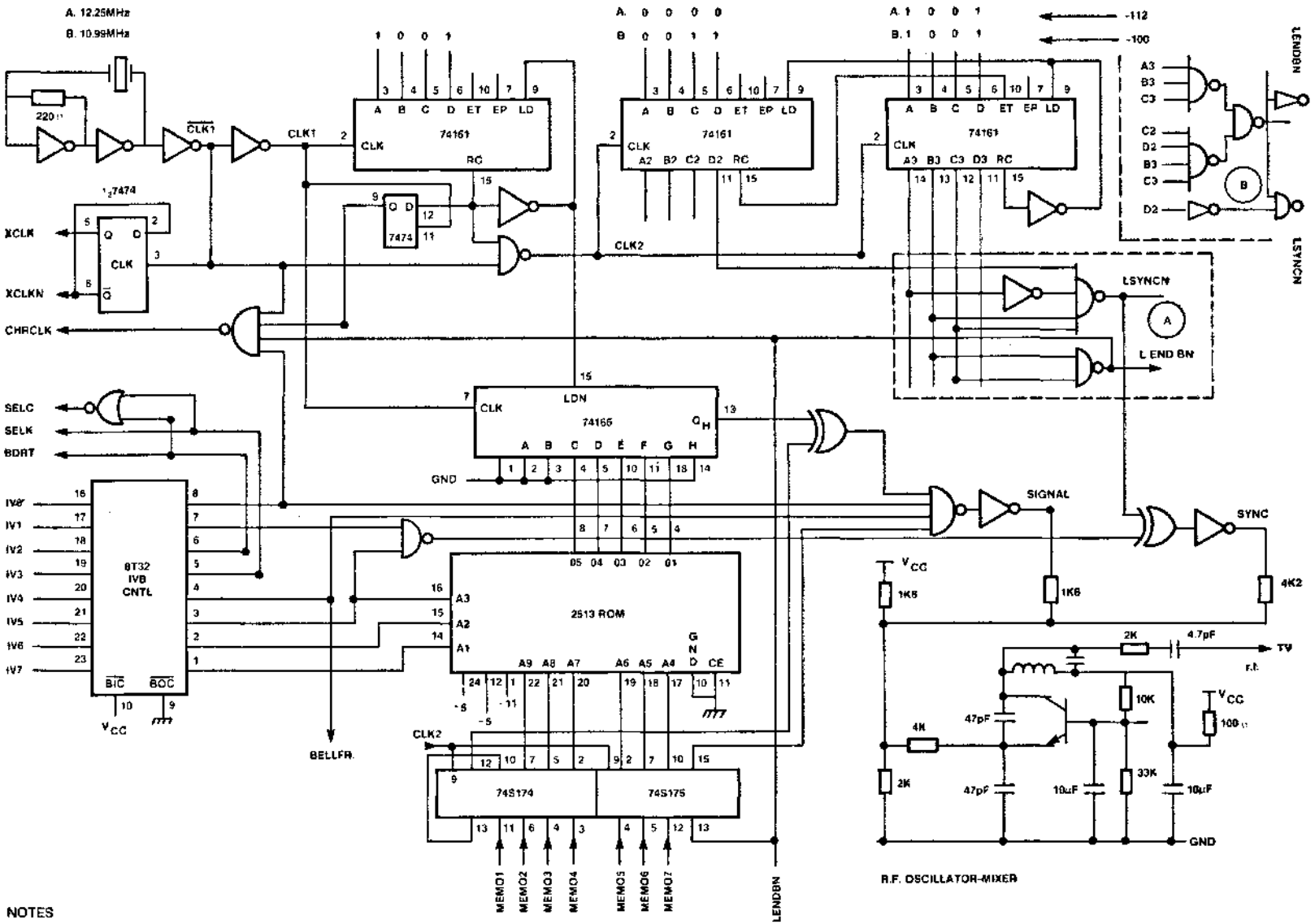
There are three ways to achieve display of 80 characters:

1. A slight change in the TV set to change the overscan to an underscan (display during 51μs).
2. Selection of the 2513 on an access time of 535ns or less.
3. Using an access overlap of the dot ROM. The address of character n+1 is then presented to the ROM inputs short time before the data of the outputs is valid for character n.

NOTE

Use of the intermediate latch enables use of the 74192 counter. This counter has a 10ns faster load operation. This 10ns makes the RAM access from the 8X300 less critical.

TIMING, CONTROL AND CHARACTER DOT GENERATION

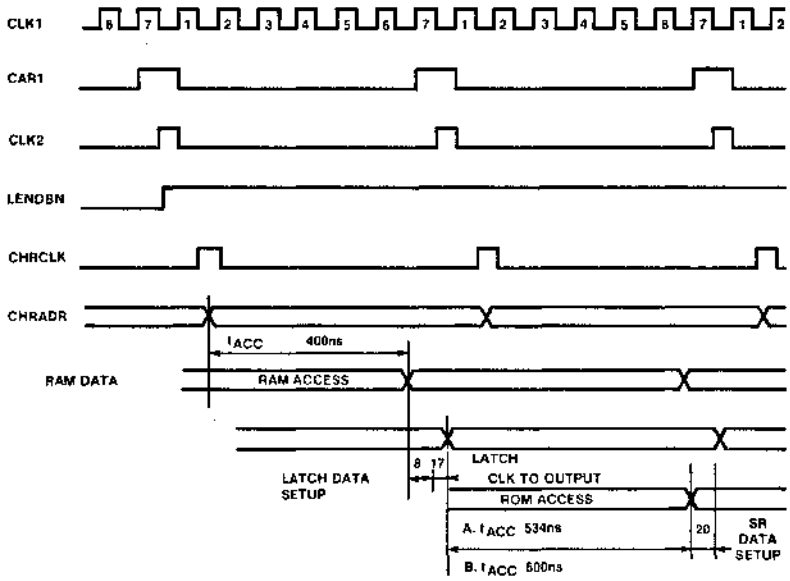


NOTES

1. 2513 access time = 535ns
2. Screen overscan changed in TV set; display time 51µs per line.

Figure 18

RAM AND ROM TIMING DIAGRAM



RAM timing		
Counter CLK to RAM address 74162		99ns
RAM address to RAM data 21L02-3		400ns
Character latch setup time		8ns
	Total	507ns (available 571ns)

ROM timing		CASE A	CASE B
Latch CLK to data out 74S174/S175		17ns	17ns
ROM address to output 2513		534ns	600ns
Shift register data setup 74166		20ns	20ns
	Total	571ns	637ns

NOTES

1. Reduced ROM access time for display of 80 characters in 45.7μs of the line
2. Current access time of 2513 ROM needs 51μs of the line.

Figure 19

TIMING, CONTROL AND CHARACTER DOT GENERATION (OVERLAPPED ROM ACCESS)

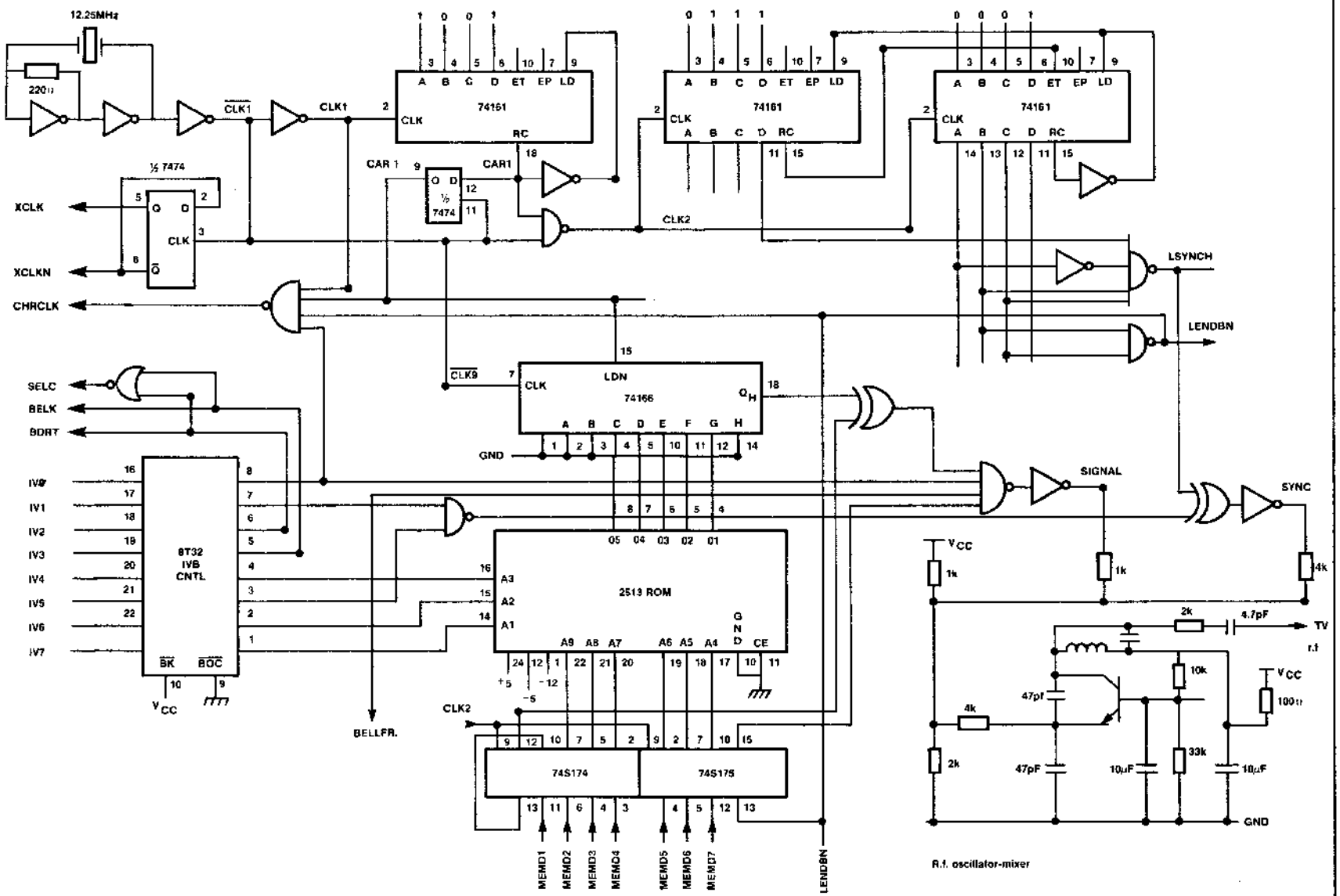
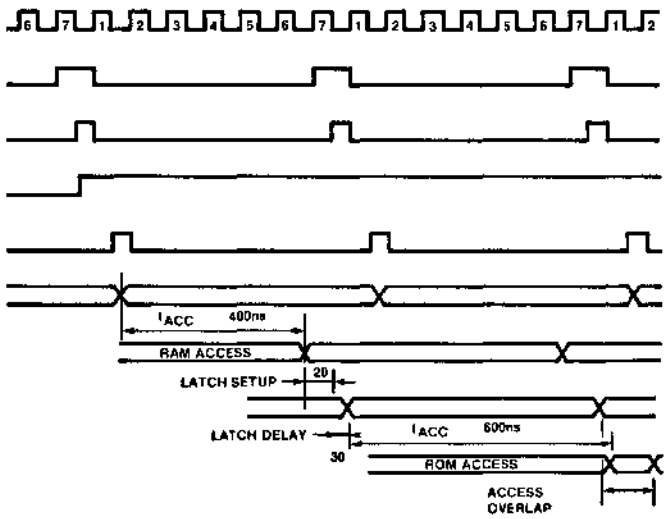


Figure 20

RAM AND ROM TIMING DIAGRAM



RAM timing		
Counter CLK to RAM address 74162		99ns
RAM address to RAM data 21L02-3		400ns
Character latch setup time 74S174/S175		8ns
	Total	507ns
ROM timing		
Latch CLK to output 74S174/S175		17ns
ROM access		600ns
Shift register data setup 74166		20ns
	Total	637ns

Available time is 8/7 X 571ns = 652ns at a character speed of 571ns (this means 80 characters in 45.7μs).
The used access overlap is then 81.5ns.

Figure 21

DESCRIPTION

The 8X300 Cross Assembly Program, MCCAP, provides a programming language which allows the user to write programs for the 8X300 in symbolic terms. MCCAP translates the user's symbolic instructions into machine-oriented binary instructions. For example, the jump instruction, JMP, to a user defined position, say ALPHA, in program storage is coded as:

JMP ALPHA

and is translated by MCCAP into the following 16-bit word (see Figure 1).

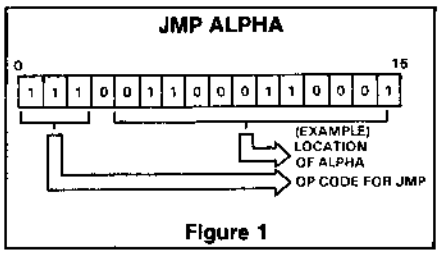


Figure 1

MCCAP allocates the 8X300 program storage and assigns Interface Vector and Working Storage address to symbols as declared in the user's program.

The ability to define data of the Interface Vector as symbolic variables is a powerful feature of MCCAP. Interface Vector variables may be operated on directly using the same instructions as those for variables in Working Storage and for the working registers.

The Assembler Declaration statements of MCCAP allow the programmer to define symbolic variable names for data elements tailored to his application. Individual bits and sequences of bits in Working Storage and on the Interface Vector may be named and operated upon directly by 8X300 instructions.

In addition to simplifying the language and bookkeeping of the program, MCCAP provides program segmentation and communication between segments; i.e., the main program and any subprograms. If a sequence of code appears more than once in a program, it can be written as a separate program segment, a subprogram, and called into execution whenever that subprogram's function is required. Program segmentation also permits the construction of a program in logically discrete units. These segments need not be written sequentially or even by the same person. The various program segments provide a function description, or block diagram, of the application. Communication between segments means that control and data can be transferred in both directions. MCCAP automatically generates the code for subprogram entry and exit mechanisms when the appropriate CALL and RTN statements are invoked.

MCCAP SOURCE PROGRAM

```
                    MICROCONTROLLER SYMBOLIC ASSEMBLER VER 1.0
1680                    *
1681                    *
1682 01544                      PROC    RDCMMD
1683                    *
1684 01544  6 07003             SEL     IVRESP          FDC RESPONSE BYTE
1685 01545  6 20101             XMIT    UR, BCTRL       ESTABLISH USER READ ONLY
1686 01546  6 07002             SEL     IVDATA          HOLDS COMMAND BYTE
1687 01547  0 27305             MOVE    FUNC, R5        FUNCTION CODE
1688 01550  0 24306             MOVE    DADDR, R6       DISK ADDRESS
1689 01551  0 21202             MOVE    BUFF, R2        BUFFER FUNCTION CODE
1690 01552  6 07003             SEL     IVRESP
1691 01553  6 25100             XMIT    0, DONE         SHOW COMMAND IN PROGRESS
1692 01554  6 20100             XMIT    UW, BCTRL       RESTORE USER WRITE
1693 01555  6 27101             XMIT    1, XFR          SIGNAL USER FDC ACCEPTED BYTE
1694 01556  6 07001             SEL     IVCTRL          USER CONTROL BYTE
1695 01557  5 26117             NZT     CMMD,*          WAIT FOR CMMD TO GO LOW
1696 01560  6 07003             SEL     IVRESP          FDC RESPONSE BYTE
1697 01561  6 27100             XMIT    0, XFR          LOWER XFR SIGNAL
1698 01562  6 07001             SEL     IVCTRL          USER CTRL BYTE
1699 01563  4 26123             XEC     *(CMMD), 2      WAIT FOR NEXT COMMAND SIGNAL
1700 01564  6 07003             SEL     IVRESP          SECOND COMMAND BYTE AVAILABLE
1701 01565  6 20101             XMIT    UR, BCTRL       SET IVDATA TO USER READ ONLY
1702 01566  6 07002             SEL     IVDATA          2ND COMMAND BYTE
1703 01567  0 27704             MOVE    TRACK, R4       TRACK ADDRESS
1704 01570  0 27503             MOVE    SECT, R3        SECTOR ADDRESS
1705 01571  6 07003             SEL     IVRESP          FDC RESPONSE BYTE
1706 01572  6 27101             XMIT    1, XFR          SIGNAL USER
1707 01573  6 20100             XMIT    UW, BCTRL       RESTORE USER WRITE
1708 01574  6 07001             SEL     IVCTRL
1709 01575  5 26135             NZT     CMMD,*          WAIT FOR CMMD TO GO LOW
1710 01576  6 07003             SEL     IVRESP          FDC RESPONSE BYTE
1711 01577  6 27100             XMIT    0, XFR          LOWER XFR SIGNAL
1712                    *
1713 01600  7 01652             RTN                     RETURN
1714                            END     RDCMMD
1715                    *
```

Figure 2

MCCAP OUTPUT

The output from a MCCAP compilation includes an assembler listing and an object module. During pass two of the assembly process, a program listing is produced. The listing displays all information pertaining to the assembled program. This includes the assembled octal instructions, the user's original source code and error messages. The listing may be used as a documentation tool through the inclusion of comments and remarks which describe the function of a particular program segment. The main purpose of the listing, however, is to convey all pertinent information about the assembled program, i.e., the memory addresses and their contents.

The object module is also produced during pass two. This is a machine-readable computer output produced on paper tape. The output module contains the specifications necessary for loading the memory of the Microcontroller Simulator (MCSIM), for loading the memory of the SMS ROM Simulator, or for producing ROMs or PROMs. The object module can be produced in MCSIM, ROM Simulator or BNPF format.

An example of a MCCAP source program is shown in Figure 2.

PROGRAM STRUCTURE

Program Segments

A MCCAP program consists of one or more program segments. Program segments are the logically discrete units, such as the main program and subprograms, which comprise a user's complete program. Program segments consist of sequences of program statements. The first program segment must be the main program. The main program names the overall program and is where execution begins. All other segments are subprograms; each subprogram must be named. Control and data can be passed in both directions between segments. No segment may call itself, or one of its callers, or the main program. Program segments take the form as shown in Figure 3.

The Assembler Declaration statements define variables and constants. They must precede the use of the declared variables and constants in the Executable Statements in a program. The Executable Statements are those which result in the generation of one or more executable machine instructions.

Subprograms

Subprograms are program segments which perform a specific function. A major reason for using subprograms is that they reduce programming and debugging labor when a specific function is required to be executed at more than one point in a program. By

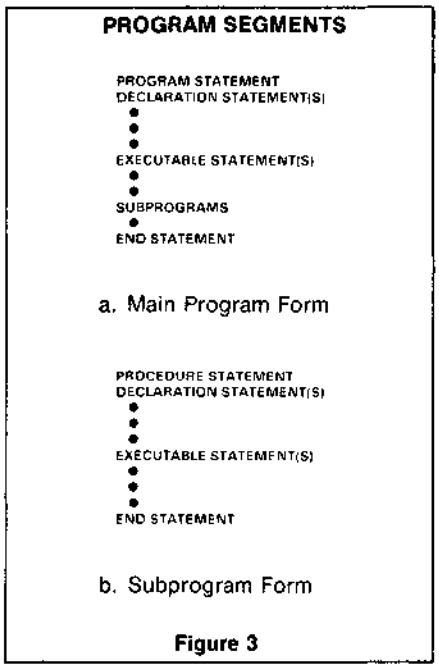


Figure 3

creating the required function as a subprogram, the statements associated with that function may be coded once and executed at many different points in a program. Figure 3 illustrates an example.

The program structure in Figure 3 causes the code associated with PROC WAIT to be executed three times within PROG MANY-WAIT. This is accomplished even though the statements associated with PROC WAIT are coded only once, rather than three times.

Subprogram Calls and Returns

For user-provided procedures, a jump to the associated procedure and a return link are created for each procedure reference. The instructions to accomplish this result in subprogram entry time. The instructions to accomplish subprogram exit result in exit time. The user may utilize the MCCAP procedure mechanism for linking calling programs with called programs or he may create his own instructions to do so. The following describes the linkage mechanism and timing for MCCAP user procedures.

Linkage between called and calling programs is achieved through the generation of an indexed "return jump" table, the length of which corresponds to the number of different times in the program that the subprograms are called. This table is generated automatically by MCCAP when procedure CALL and RTN statements are invoked. For each procedure reference, MCCAP creates two statements in the calling program. Thus, the time required for the subprogram entry is 0.5 microseconds. The subprogram return mechanism requires the execution of two instructions or 0.5 microseconds. These times do not include saving and restoring of the working registers. The total time to save all working registers is 3.5 microseconds, the same time to restore all registers. Saving of all working registers is normally not necessary, but worst case calculations for entry and exit time below do include this time. Thus, subprogram exit and entry times are:

$$0.5\mu s \leqslant \text{Entry Time} \leqslant 4.0\mu s$$
$$0.5\mu s \leqslant \text{Exit Time} \leqslant 4.0\mu s$$

Details of the code required for procedure CALL and RTN are provided in the Programming Examples section. See Figures 21 and 22.

Macros

A macro is a sequence of instructions that can be inserted in the assembly source text by encoding a single instruction. The macro is defined only once and may then be invoked any number of times in the program. This facility simplifies the coding of programs, reduces the chance of errors, and makes programs easier to change.

A macro definition consists of a heading, a body and a terminator. This definition must precede any call on the macro. In MCCAP, the heading consists of the MACRO statement which marks the beginning of the macro and names it. The body of the macro is made up of those MCCAP statements which will be inserted into the source code in place of the macro call. The terminator consists of an ENDM statement which marks the physical end of the macro definition.

MCCAP Statements

The MCCAP language consists of thirty statements categorized as follows:

Assembler Directive Statements
Assembler Declaration Statements
Communication Statements
Macro Statements
Machine Statements

The following lists the statements in each category, describes their use, and provides examples. Detailed use of the instructions including rules of syntax and parameter restrictions are described in the MCCAP Reference Manual.

Assembler Directive Statements

Assembler Directive statements define program structure and control the assembler outputs. They do not result in the generation of 8X300 executable code. There are twelve Assembler Directive statements:

PROG Statement
PROC Statement
ENTRY Statement
END Statement
ORG Statement
OBJ Statement
IF Statement
ENDIF Statement
LIST Statement
NLIST Statement
EJCT Statement
SPAC Statement

PROG Statement

Use
Defines the names and marks the beginning of a main program.

Example: PROG PROCESS

PROC Statement

Use
Defines the names and marks the beginning of a subprogram.

Example: PROC WAIT

ENTRY Statement

Use
Defines the name and marks the location of a secondary entry point to a subprogram.

Example: ENTRY POINT 2

END Statement

Use
Terminates a program segment or a complete program.

Examples: END SUB1
END MAIN

ORG Statement

Use
Sets the program counter to the value specified in the operand field.

Example: ORG 200

OBJ Statement

Use
To specify the format of the object module.

Examples: OBJ R
OBJ M
OBJ N

NOTE

"R" indicates the ROM Simulator format. "M" indicates the Microcontroller Simulator format. "N" indicates BNPF format.

IF Statement

Use
To mark the beginning of a sequence of code, which may or may not be assembled depending on the value of an expression.

Examples: IF VAL
IF X + Y

MOVE Statement

Use

To copy the contents of a specified register, WS variable or IV variable into a specified register, WS or IV. Defined in Instruction Descriptions.

Examples:
```
MOVE R1(6),R6
MOVE X,Y
```

NOTE

The first example illustrates a six place right rotate of R1's data before it is moved to R6. The contents of R1 are not affected. The second example may be a Working Storage or Interface Vector variable move, depending on the way X and Y are defined in Declaration Statements.

ADD Statement

Use

To add the contents of a specified register, WS variable, or IV variable to the contents of the AUX register and place the result in a specified register, WS variable or IV variable.

Examples:
```
ADD R1(3),R2
ADD DATA,OUTPUT
```

NOTE

The first example illustrates a three place right rotate of R1's data before the addition is carried out. Under certain conditions a rotate may be used to multiply the specified operand by a power of 2 before the addition is done. The contents of R1 are not affected. The second example suggests that the contents of WS variable have been added to the contents of the AUX register and the result placed in an IV variable, making the result immediately available to the user's system.

AND Statement

Use

To compute the logical AND of the contents of a specified register, WS variable or IV variable and the contents of the AUX register. The logical result is placed in a specified register, WS variable or IV variable. In actual practice, the AND statement is often used to mask out undesired bits of a register.

Examples:
```
AND R2,R2
AND R3(1),R5
AND X,Y
```

NOTE

The first example illustrates the use of an AND statement in what might be a masking operation. If the AUX register contains 00001111 then this statement sets the 4 high order bits of R2 to 0 no matter what they were originally. The 4 low order bits of R2 would be unaffected.

The second example illustrates a one place rotate to the right of R3's data before the AND is carried out. The contents of R3 are not affected. In the third example, X and Y may be parts of the same WS or IV byte, or one may be a WS byte and the other an IV byte.

XOR Statement

Use

To compute the logical exclusive OR of the contents of a specified register, WS variable or IV variable and the contents of the AUX register, and place the result in a specified register, WS variable or IV variable. In practice, the XOR statement is often used to complement a value and to perform comparisons.

Examples:
```
XOR R6,R11
XOR R1(7),R4
XOR X,Y
```

NOTE

The first example illustrates the use of an XOR statement in what might be a complementing operation. If the AUX register contains all 1's then the execution of this statement results in the complement of the contents of R6 replacing the contents of R11. The second and third examples are of the same form as the second and third examples of the AND statement.

XMIT Statement

Use

To transmit or load literal values into registers, WS variables or IV variables.

Examples:
```
XMIT DATA,IVR
XMIT OUTPUT,IVL
XMIT -11,AUX
XMIT -00001011B,AUX
XMIT -13H,AUX
```

NOTE

The first example selects a previously declared WS variable by transmitting its address to the IVR register. The second example selects a previously declared IV variable by transmitting its address to the IVL register. The last three examples all result in the generation of the same machine code. They all load the AUX register with -11_{10}. In the first case, the programmer has written the number in base 10. In the second case, the programmer has written the number in binary and has indicated this by placing a B after the number. In the third case, the number has been written in octal as indicated by an H after the number.

XEC Statement

Use

To select and execute one instruction out of a list of instructions in program memory as determined by the value of a data variable, and then continue the sequential execution of the program beginning with the statement immediately following the XEC unless the selected instruction is a JMP or NZT statement.

Examples:
```
            XEC JTABLE(R1),3
JTABLE  JMP GR8ERTHAN
        JMP LESSTHAN
        JMP EQUALTO

            XEC SEND(INPUT),4
            "NEXT INSTRUCTION"
            "NEXT INSTRUCTION"
             .       .
             .       .
SEND        XMIT 11011011B,AUX
            XMIT 11111111B,AUX
            XMIT 10101010B,AUX
            XMIT 00000000B,AUX
```

NOTE

In the first example, the execution of the program will transferred to one of three labeled instructions on the basis of whether register R1 contains 0, 1 or 2. In the second example, the XEC statement causes the execution of a statement which transmits a special bit pattern to the AUX register in response to an input signal which is either 0, 1, 2 or 3. After the pattern is transmitted, the execution of the program continues with the next instruction after the XEC.

NZT Statement

Use

To carry out a conditional branch on the basis of whether or not a register, WS variable, or IV variable is zero or non-zero.

Examples:
```
NZT R1,*+2
NZT SIGN,NEG
```

NOTE

In the first example, if the contents of R1 are non-zero, then program execution will continue with the instruction, whose address is the sum of the address of the NZT statement and 2. If the contents of R1 are 0, the program execution continues with the next instruction after the NZT statement. In the second example, if the contents of a WS or IV variable called SIGN is non-zero, then program execution will continue beginning with the instruction whose address is NEG. Otherwise execution continues with the next instruction after the NZT statement.

JMP Statement

Use

To transfer execution of the program to the statement whose address is the operand of the JMP statement.

Examples:
```
JMP START
JMP *-2
```

NOTE

In the first example, execution of the program continues sequentially beginning with the instruction labeled START. In the second example, program execution continues beginning with the instruction whose address is the JMP instruction's address minus 2.

SEL Statement

Use

Select a variable in Working Storage or on the Interface Vector, so that subsequent machine instructions may reference that variable.

Examples:
```
SEL DATA
SEL OUTPUT
```

NOTE

It is the programmer's responsibility to assure that the proper page has been addressed before calling the SEL statement if the variable may be in Working Storage. The SEL statement causes a single instruction, XMIT, to be assembled into the user program. The operand of the XMIT instruction is the byte address of the named variable (argument of the reference) as it has been allocated in Working Storage or on the Interface Vector.

PROGRAMMING EXAMPLES

This section contains programming examples which demonstrate how the 8X300's instructions can be assembled to perform some simple, commonly required functions. These examples are written as program

fragments. They are not complete programs as the Data Declaration and Directive statements have been omitted. Otherwise, they follow standard MCCAP conventions.

Looping

Looping is terminated by incrementing a counter and testing for zero. Register R1 is used as counter register and is loaded with a negative number so that the program counts up to zero. Figure 4 illustrates the process.

LOOPING

```
        XMIT    NEG,R1      Load negative loop count.
ALPHA   ...
        .
        .
        .                   Loop start.
        XMIT    1,AUX       Store increment value in AUX register
                            which is an implicit operand of ADD
                            instruction.
        ADD     R1,R1       Increment counter register. Add con-
                            tents of AUX to contents of R1 and
                            store the sum in R1.
        NZT     R1,ALPHA    Test contents of R1 for zero. If zero,
                            execute next sequential instruction,
                            otherwise, jump to ALPHA and contin-
                            ue execution from there.
        .
        .
        .
```

TIME: 750 nanoseconds

Figure 4

Inclusive-OR (8 Bits)

Generate inclusive-OR of the contents of R1 and R2. Store the logical result in R3. Although the 8X300 does not have an OR instruction, it can be quickly implemented by making use of the fact that (A + B) + (A B) is logically equivalent to A ⊕ B.

INCLUSIVE-OR

```
MOVE    R2,AUX      Load one of the operands into AUX
                    register so that it can be used as the
                    implicit operand of XOR and AND
                    instructions.
XOR     R1,R3       Take exclusive OR of AUX and R1.
                    Store result in R3.
AND     R1,AUX      Take AND of AUX and R1. Place re-
                    sults in AUX.
XOR     R3,R3       Take exclusive OR of AUX (A · B) and
                    R3 (A + B). Store result in R3. R3 now
                    contains inclusive OR of R1 and R2.
```

TIME: 1.0 microseconds

Figure 5

Two's Complement (8-Bits)

Generate the two's complement of the contents of R2. Store the result in R3. Assume that R2 does not contain 200_8.

TWO'S COMPLEMENT

```
XMIT    -1,AUX      Load AUX in preparation for XOR.
XOR     R2,R3       1's complement of R2 is now in R3.
XMIT    1,AUX       Load AUX in preparation for ADD.
ADD     R3,R3       2's complement of R2 is now in R3.
```

TIME: 1.0 microseconds

Figure 6

8-Bit Subtract

Subtract the contents of R2 from the contents of R1 by taking the two's complement of R2 and adding R1. Store the difference in R3.

8-BIT SUBTRACT

```
XMIT    -1,AUX      Perform 2's complement, R2.
XOR     R2,R3
XMIT    1,AUX
ADD     R3,AUX      2's complement of R2 is now in AUX.
ADD     R1,R3       R1-R2 is now in R3.
```

TIME: 1.25 microseconds

Figure 7

16-Bit ADD, Register to Register

Add a 16-bit value stored in R1 and R2 to a 16-bit value in R3 and R4. Store the result in R1 and R2.

16-BIT ADD, REGISTER TO REGISTER

```
MOVE    R2,AUX      Move low order byte of first operand
                    to AUX in preparation for ADD.
ADD     R4,R2       Add the low order bytes of the two
                    operands and store the result in R2.
                    R2 contains the low order byte of the
                    result.
MOVE    R1,AUX      Move high order byte of first operand
                    to AUX.
ADD     OVF,AUX     Add in possible carry from addition of
                    low order bytes.
ADD     R3,R1       Add the high order bytes plus carry
                    and place result in R1. R1 contains the
                    high order byte of the result.
```

TIME: 1.25 microseconds

Figure 8

16-Bit ADD, Memory to Memory

Add a 16-bit value in Working Storage, OPERAND1, to a 16-bit value in Working Storage, OPERAND2, and store result in Working Storage OPERAND1. H1 and L1 represent the high and low order of bytes OPERAND1. H2 and L2 represent the high and low order bytes of OPERAND2.

16-BIT ADD, MEMORY TO MEMORY

```
XMIT    L2,IVR      Transmit address of low order byte of
                    second operand to IVR.
MOVE    L2,AUX      Move low order byte to AUX.
XMIT    L1,IVR      Transmit address of low order byte of
                    first operand to IVR.
ADD     L1,L1       Add low order bytes and store result in
                    L1.
MOVE    OVF,AUX     Move possible carry from addition of
                    low order bytes to AUX register.
XMIT    H2,IVR      Add high order byte of second oper-
                    and to possible carry. Store result in
                    AUX.
ADD     H2,AUX
XMIT    H1,IVR
ADD     H1,H1       High order byte of sum is in H1. Low
                    order byte of sum is in L1.
```

TIME: 2.25 microseconds

Figure 9

Byte Assembly From Bit Serial Input

This is typical of problems associated with interfacing to serial communications lines. An 8-bit byte is assembled from bit inputs that arrive sequentially at the Interface Vector. A single bit on the Interface Vector named STROBE is used to define bit timing, and a second bit, named INPBIT, is used as the bit data interface. Figure 10 illustrates the byte assembly.

BYTE ASSEMBLY

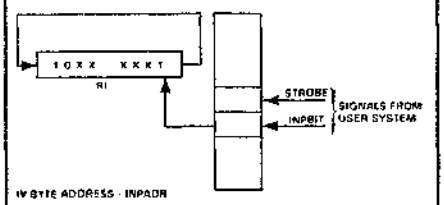


Figure 10

BYTE ASSEMBLY PROGRAM

```
XMIT    0,R1        R1 will be used as a character
                    buffer. It has been cleared.
XMIT    8,R2        R2 will be used as a bit counter.
XMIT    INPADR,IVL  Select IV Byte that contains
                    INPBIT and STROBE.
NEXT BIT N2T STROBE,*+2 Test STROBE for data ready.
                    The MOVE instruction is exe-
                    cuted only when STROBE = 1.
JMP     *-1         Loop until STROBE = 1.
MOVE    INPBIT,AUX
XOR     R1(1),R1    Rotate R1 one place right. This
                    puts a zero in the least signifi-
                    cant bit position. Then take the
                    exclusive OR of this rotated
                    version of R1 and of AUX. Place
                    the result in R1. The least sig-
                    nificant bit of R1 will now equal
                    the latest value of INPBIT.
XMIT    -1,AUX
ADD     R2,R2       Decrement R2.
                    If R2 is not yet zero, then more
                    bits must be collected to com-
                    plete the byte being assembled.
MOVE    R1(1),R1    This instruction will only be
                    executed when 8 bits have been
                    collected. After this is done, it is
                    still necessary to rotate one
                    more time to get the last INP-
                    BIT into the high order bit posi-
                    tion of R1.
```

TIME: 1.8 microseconds per bit (minimum)

Figure 11

Rotate Left

The 8X300 has no instructions which explicitly rotate data to the left. Such an instruction would be redundant because of the circular nature of the rotate operation. For example, a rotate of two places to the left is identical to a rotate of six places to the right. The rotate n places to the left in an 8-bit register, rotate 8-n places to the right. This example illustrates a rotate of the contents of R4 three places to the left.

MOVE R4(5),R4
TIME: 250 nanoseconds

Three Way Compare

The contents of R1 are compared to the contents of R2. A branch is taken to one of three points in the program depending upon whether R1 = R2, R1 < R2, or R1 > R2.

THREE WAY COMPARE

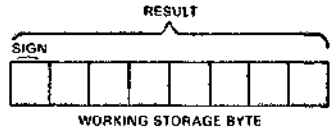


Figure 12

THREE WAY COMPARE PROGRAM

```
XMIT    RESULT,IVR      Choose a working Storage byte
                        by transmitting its address to
                        IVR register.
XMIT    -1,AUX          Load AUX with all 1's, in prep-
                        aration for complementing con-
                        tents of R2
XOR     R2,RESULT       Store complement of R2 in RE-
                        SULT.
XMIT    1,AUX
ADD     RESULT,AUX      AUX now contains 2's comple-
                        ment of R2.
ADD     R1,RESULT       RESULT now contains R1-R2.
NZT     RESULT,NEQUAL   If RESULT ≠ 0, then R1 ≠ R2.
JMP     EQUAL
NEQUAL  NZT SIGN,LESS   Sign Bit = 1 only when R1 < R2.
GREATER Continue
                          •
                          •
                          •
EQUAL   Continue
                          •
                          •
                          •
LESS    Continue
TIME: 2.0 microseconds
```

Figure 13

Interrupt Polling

Three external interrupt signals are connected to three IV bits. The three bits are scanned by the program to determine the presence of an interrupt request. A branch is taken to one of eight program locations depending upon whether any or all of the interrupt request signals are present. The IV bits associated with the interrupt requests are wired to the low order three bits of the IV byte named Control. Figures 14 and 15 illustrate the interrupt polling.

INTERRUPT POLLING

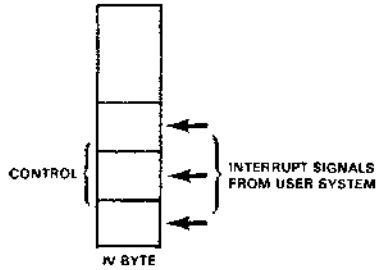


Figure 14

INTERRUPT POLLING PROGRAM

```
XMIT    CONTROL,IVL     Choose proper IV Byte
                        by transmitting its ad-
                        dress to IVL register.
XEC     JTABLE (CONTROL),8  Execute the one in-
                        struction whose ad-
                        dress is the sum of
                        JTABLE and the con-
                        tents of CONTROL. The
                        8 indicates the length of
                        the table.
                          •
                          •
                          •
JTABLE  JMP   ALPHA1    Table of 8 instructions,
                        one of which is execut-
                        ed as a result of the XEC
                        instruction above.
        JMP   ALPHA2
          •
          •
          •
        JMP   ALPHA8
TIME: 750 nanoseconds.
```

Figure 15

Bit Pattern Detection In An I/O Field

Test input field called Input for specific bit pattern, for example: 1 0 1 1. If pattern is not found, branch to NFOUND, otherwise continue sequential execution. Figures 16 and 17 illustrate the procedure.

BIT PATTERN DETECTION

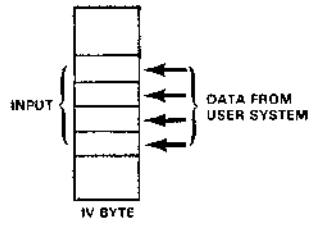


Figure 16

BIT PATTERN DETECTION PROGRAM

```
XMIT    INPUT,IVL       Choose proper IV Byte by trans-
                        mitting its address to IVL regis-
                        ter.
XMIT    1011B,AUX       Store desired bit pattern in AUX
                        register for use as implicit oper-
                        and of XOR instruction.
XOR     INPUT,AUX       Take exclusive OR of the con-
                        tents of INPUT and AUX. Store
                        the result in AUX. Now the con-
                        tents of AUX will be zero if the
                        contents of INPUT are 1011.
NZT     AUX,NFOUND      Test AUX for zero. Branch to
                        NFOUND if non-zero.
  •  •
  •  •
  •  •
NFOUND Continue
TIME: 1.0 microseconds
```

Figure 17

Control Sequence #1

Set an output bit when an input bit goes high (is set) (see Figure 18).

CONTROL SEQUENCE #1

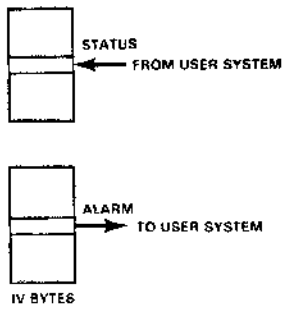


Figure 18

CONTROL SEQUENCE #1 PROGRAM

```
XMIT    STATUS,IVL      Choose input IV byte by trans-
                        mitting its address to IVL.
NZT     STATUS,*+2      Test input bit to determine
                        whether it is still zero. Skip next
                        instruction if it is not zero
JMP     *-1             Jump to previous instruction.
XMIT    ALARM,IVL       Choose output IV byte.
XMIT    1,ALARM         Set output bit by loading ALARM
                        with 1.
TIME: 1.0 microseconds (minimum)
```

Figure 19

Control Sequence #2

Output a specific 5-bit pattern in response to a specified 3-bit input field.

Subprogram Calls and Returns

The mechanism for managing subprogram calls and returns is based on assigning a return link value to each subprogram caller; this return link value is then used, on exit from the subprogram, to index into the return jump table which returns control to the callers of the subprogram. Figure 21 is an example of a subprogram called from four different locations in the main program.

As seen from Figure 21, each subprogram (or procedure) caller is assigned a "tag" or index values ranging from 0 to 3, or a total of four index values for the four callers. Before jumping to the subprogram, the index value is placed in a previously agreed upon location, register R11 in this case. Upon exit from the subroutine, the index value stored in R11 is used as an offset to the Program Counter in order to execute the proper JMP instruction. The key to returning to the proper caller is the index jump table. Figure 22 gives a detailed description of the return operation.

CONTROL SEQUENCE #2 PROGRAM

```
        XMIT   STATUS,IVL     Choose the IV byte which receives the 3-bit input from user's system
        MOVE   STATUS,R1      Move the 3 bits of interest from the IV byte to register R1. The 3 bits are automatically right justified.
        XMIT   ALARM,IVL      Choose the IV byte through which the response is sent to the user's system.
        XEC    PATTERN(R1),8  Select specific pattern from PATTERN table.
        JMP    *+9
PATTERN XMIT   A,ALARM
        XMIT   B,ALARM
        XMIT   C,ALARM
        •      •
        •      •              Transmit proper pattern to output IV byte subfield by executing just one
        •      •              of these eight instructions. A through H represent the names associated
        XMIT   H,ALARM        with eight different control bit patterns.
        •      •
        •      •
        •      •
```

TIME: 1.25 microseconds.

Figure 20

RETURN OPERATION

Address n	XEC*+1	This instruction results in the execution of the instruction located at the current value of the Program counter p plus 1 plus the contents of R11, which is the caller index value.
Address n+1	JMP A	
Address n+2	JMP B	The JMP table follows in consecutive Program Storage locations following XEC.
Address n+3	JMP C	
Address n+4	JMP D	

Figure 21

SUBPROGRAM CALL

```
Program
Storage
Address                          Instruction
000137₈  XMIT 0, R11     Load AUX with 0Caller # 1
000140₈  JMP SUBR        Jump to start of subprogram.
000141₈  Next Instruction
         •
         •
         •
         •
001133₈  XMIT 1, R11     Load AUX with 1 Caller # 2
001134₈  JMP SUBR        Jump to start of subprogram.
001135₈  Next Instruction
         •
         • •
         •
003260₈  XMIT 2, R11     Load AUX with 2 Caller # 3
003261₈  JMP SUBR        Jump to start of subprogram
003262₈  Next Instruction
         •
         •
         •
         •
003654₈  XMIT 3, R11     Load AUX with 3Caller #4
003655₈  JMP SUBR        Jump to start of subprogram
003656₈  Next Instruction
         •
         •
         •
         •
```

a. Main Program

```
SUBR     Machine Instructions
         JMP  TABL
         •
         •
         •
         •

Subroutine
Return
Code

TABL     XEC  *+1(R11)   Execute JMP located at
                         current PC + 1 + (R11).
         JMP  000141₈    Return to Caller #1
         JMP  001135₈    Return to Caller #2
         JMP  003262₈    Return to Caller #3
         JMP  003656₈    Return to Caller #4
```

b. Subroutine

Figure 22

INTRODUCTION

The MicroController Program Library is made up of commonly required subprograms in the form of procedures and the documentation for each of these procedures. The Library is intended to support applications for the MicroController. These applications may be written in the MicroController Cross Assembler (MCCAP) language, a stand-alone ANSI FORTRAN assembler intended to execute on any 16-bit or larger minicomputer.

SPECIAL RESTRICTIONS WHEN USING THE CROSS ASSEMBLER

Where IV Bytes and Working Storage allocations are required by a procedure, they have been made. These arbitrary definitions should be reviewed by the procedure user and modified as necessary to fit his application.

There are two characteristics of the library procedures that require special attention by the user. First of all, procedures from the library may in turn call on other library procedures. For example, a procedure which generates square roots would make calls on a division and possibly a multiplication procedure. These calls, if any, are noted in the documentation for each procedure. The programmer must be aware of these calls because he must append these secondary procedures to his program.

The second characteristic is that the Cross Assembler has two instructions (XEC, NZT) whose operands are required to be within the same register page and/or memory page. It is difficult, if not impossible, to determine before compile time whether or not these ranges will be exceeded. The solution is to code conditional ORG statements which, when properly used, align the program with the next page boundary only when necessary. Conditional ORG statements have been included in all the library procedures. These statements should prevent most, but not necessarily all, range errors within the procedure. The customer may find it necessary to encode additional ORG statements in some instances. The user should be aware that this solution creates blocks of unused memory in program storage. The programmer should be aware of these blocks because it is usually possible to reorganize code to either eliminate them completely or at least keep their size to a minimum.

PROCEDURE NAME: TAD16

General Description:

TAD16 is a double precision (16-bit) 2's complement addition program which checks for arithmetic overflow by comparing the signs of the operands and the result. Overflow has occurred when and only when the operands have like signs and the result has the opposite sign. When overflow occurs the program returns the value 100000 base 8. This is the largest negative 16-bit 2's complement number. TAD16 requires that its two double precision operands always be found in the same four memory locations. These four locations can be anywhere on page 0 of working storage and do not have to be contiguous. All results are stored in the two working storage locations which originally held the second operand. See Figure 1 for the flow chart and Figure 2 for the program listing.

Memory Requirements:

Program Memory: 24 words
Working Storage: 4 bytes

Registers Used and Their Logical Function:

R1 This register is used to hold information on the signs of the operands. R1 contains 0 if both operands are position, 2 if both operands are negative, or, 1 if the operands have opposite signs.

Symbols:

AL Low order byte of A
AH High order byte of A
A1 High order (sign) bit of A
BL Low order byte of B
BH High order byte of B
B1 High order (sign) bit of B

Timing:

Worst Case: 5.25 microseconds when overflow occurs
Best Case: 3.75 microseconds when operands have opposite signs

Calls On Other Library Procedures: None

TAD16 PROGRAM LISTING

```
                PROC TAD16
                MCLIB PROCEDURE TO ADD TWO 16 BIT NUMBERS
                In 2's COMPLEMENT NOTATION AND CHECK
                FOR ARITHMETIC OVERLFLOW.
                DATA DECLARATIONS.
AL RIV 200,7,6
AH RIV 201,7,6
A1 RIV 201,0,1
BL RIV 202,7,6
BH RIV 203,7,6
B1 RIV 203,0,1
TAD             SEL     A1
                MOVE    A1,AUX
                SEL     B1
                ADD     B1,R1           R1 = 0 IF BOTH = 0.
                                        R1 = 2 IF BOTH = 1.
                                        R1 = 1 IF BOTH DIFF.
                SEL     AL
                MOVE    AL,AUX
                SEL     BL
                ADD     BL,BL           AL + BL NOW IN BL.
                MOVE    OVF,AUX
                SEL     AH
                ADD     AH,AUX
                SEL     BH
                ADD     BH,BH           ANSWER IN BH . . BL
                ORG     4,256
                XEC     ** + 1(R1)
                JMP     ZEROS
                JMP     INBOUND$
                JMP     ONES
                ORG     4,32
ZEROS           NZT     B1,OVERFLOW
                JMP     INBOUNDS
                ORG     5,32
ONES            NZT     B1,INBOUNDS
OVERFLOW        XMIT    200H,AUX
                MOVE    AUX,BH
                SEL     BL
                XMIT    0,BL
INBOUNDS        RTN
                END     TAD16
```

Figure 2

TAD16 FLOW CHART

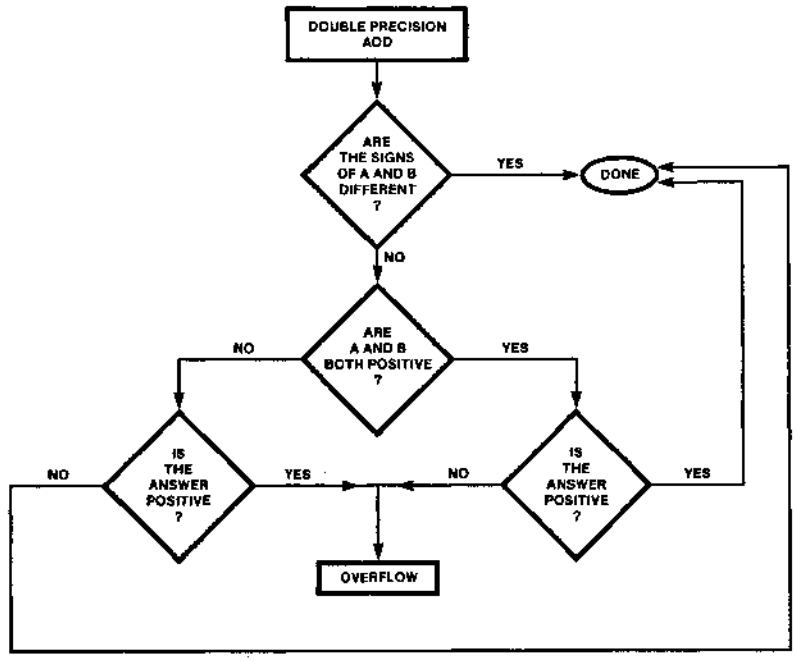


Figure 1

PROCEDURE NAME: MUL8X8

General Description:

MUL8X8 is a procedure which multiplies two 8-bit 2's complement numbers. For reasons of speed, negative numbers are converted to positive numbers before the multiplication takes place. Afterward, the product is given the proper sign. The algorithm is a straight forward add and shift routine. The operands are taken from R1 and R2. The low order byte of the sixteen bit result is stored in R1. The high order byte is stored in R3. See Figure 3 for the flow chart and Figure 4 for the program listing.

Memory Requirements:

Program Storage: 47 words
Working Storage: None

Registers Used And Their Logical Function:

R1 Initially contains the multiplier. Eventually contains the low order byte of the product.
R2 Initially contains the multiplicand.
R3 Contains the high order byte of the result.
R4 Bit counter.
R5 Contains information on the sign of the result. R5=0 if the result is negative or R5=1 if the result is positive.

Timing:

Worst Case: 35.75 microseconds

Calls On Other Library Procedures: None

MUL8X8 FLOW CHART

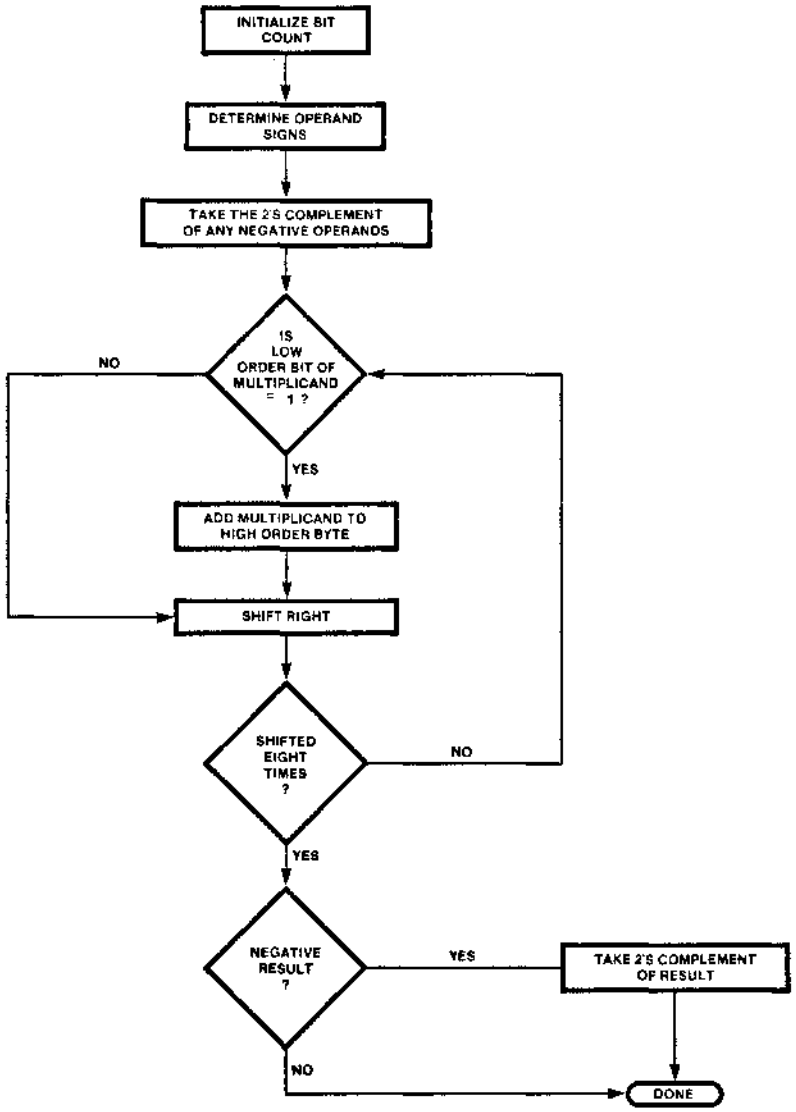


Figure 3

MUL8X8 PROGRAM LISTING

```
        MCLIB PROCEDURE TO MULTIPLY TWO 8 BIT 2's
        COMPLEMENT NUMBERS TO GENERATE A
        16-BIT RESULT.
        XMIT  0,R3
        XMIT  -8,R4
        XMIT  1, AUX        ACCESS/TEST OPERAND1 SIGN.
        AND   R1 (7),AUX
        ORG   5,856
        XEC   TAB (AUX)
        NZT   AUX,COMP1
        JMP   OP2
TAB     XMIT  1,R5          POSITIVE.
        XMIT  0,R5          NEGATIVE.
COMP1   XMIT  377H,AUX      COMP OPERAND1.
        XOR   R1,R1
        XMIT  1,AUX
        ADD   R1,R1
OP2     XMIT  1,AUX         ACCESS/TEST OPERAND2 SIGN.
        AND   R2(7),AUX
        XOR   R5,R5
        NZT   AUX, COMP2
        JMP   LOOP
COMP2   XMIT  377H, AUX     COMP OPERAND2.
        XOR   R2,R2
        XMIT  1,AUX
        ADD   R2,R2
        ORG   16,256
LOOP    XMIT  1,AUX         LOW ORDER BIT 1?
        AND   R1,AUX
        NZT   AUX,* + 2
        JMP   SHIFT
        MOVE  R2,AUX        YES, ADD MULTIPLICAND.
        ADD   R3,R3
SHIFT   XMIT  177H,AUX      SHIFT PARTIAL
        AND   R1(1),R1      PRODUCT RIGHT.
        XMIT  200H,AUX
        AND   R3,(1), AUX
        XOR   R1,R1
        XMIT  177H,AUX
        AND   R3(1),R3
        XMIT  1,AUX
        ADD   R4,R4
        NZT   R4, LOOP      DONE?
        ORG   9,256
        NZT   R5,ENDD
        XMIT  377H,AUX      NO. 2's COMP.
        XOR   R1,R1
        XOR   R3,R3
        XMIT  1,AUX
        ADD   R1,R1
        MOVE  OVF,AUX
        ADD   R3,R3
ENDD    RTN
        END   MUL8X8
```

Figure 4

PROCEDURE: M16X16

General Description:

M16X16 is a procedure which multiplies two 16-bit 2's complement numbers to generate a 32-bit result. This is accomplished by adding the multiplicand to the two high order bytes of the partial product when the low order bit of the multiplier is 1 and then shifting the partial product and the multiplier one place to the right. If the low order bit of the multiplier is zero, then you shift only. This procedure requires the use of all eight internal registers. M16X16 uses the contents of R3, R4 as the multiplier and the contents of R5, R6 as the multiplicand. It is the responsibility of the calling program to set up these operands and to save the contents of R1, R2, and R11 if necessary. See Figure 5 for the flow chart and Figure 6 for the program listing.

Memory Requirements:

Program Storage: 76 words
Working Storage: 9 bits

Registers Used And Their Logical Functions:

R1 Fourth (high order) byte of result.
R2 Third byte of result.
R3 Originally the high order byte of the multiplier. Finally the second byte of the result.
R4 Originally the low order byte of the multiplier. Finally the first (low order) byte of the result.
R5 High order byte of the multiplicand.
R6 Low order byte of the multiplicand.
R11 Bit counter used to terminate the procedure.

Timing:

Average if both operands are negative: 112.5μs
Average if both operands are positive: 109.5μs
Worst case: 122.0μs

Calls On Other Library Procedures: None

M16X16 FLOW CHART

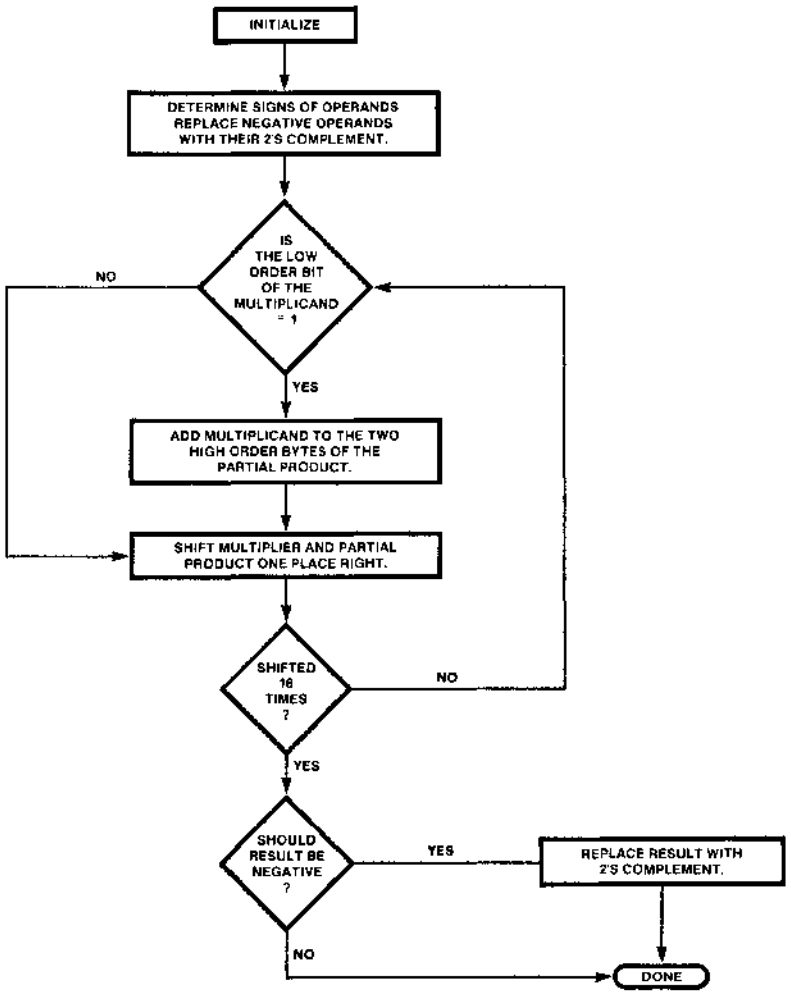


Figure 5

M16X16 PROGRAM LISTING

```
                PROC M16X16
                MCLIB PROCEDURE TO MULITPLY TWO
                16-BIT 2's COMPLEMENT NUMBERS.

                LINK RIV 200,7,8
                SRESULT PIV 201,7,1

                INITIALIZE
        SEL     LINK   SAVE LINK.
        MOVE    R11,LINK
        SEL     SRESULT
        XMIT    0,R1
        XMIT    0,R2
        XMIT    -16,R11
        DETERMINE SIGNS OF THE
        OPERANDS. REPLACE NEG.
        OPERANDS WITH THEIR 2's
        COMPLEMENT.
OP1     XMIT    1,AUX
        AND     R3(7),AUX
        ORG     6,256
        XEC     TAB(AUX)
        NZT     AUX,COMP1
        JMP     OP2
TAB     XMIT    1,SRESULT
        XMIT    0,SRESULT
COMP1   XMIT    377H,AUX
        XOR     R4,R4
        XOR     R3,R3
        XMIT    1,AUX
        ADD     R4,R4
        MOVE    OVF,AUX
        ADD     R3,R3
OP2     XMIT    1,AUX
        AND     R5(7),AUX
        XOR     SRESULT,SRESULT
        NZT     AUX,COMP2
        JMP     LOOP
COMP2   XMIT    377H,AUX
        XOR     R6,R6
        XOR     R5,R5
        XMIT    1,AUX
        ADD     R6,R6
        MOVE    OVF,AUX
        ADD     R5,R5
        ADD MULTIPLICAND TO THE TWO HIGH
        ORDER BYTES OF THE PARTIAL
        PRODUCT IF THE LOW ORDER BIT OF
        THE MULTIPLIER IS 1.
        ORG     29,256
LOOP    XMIT    1,AUX
        AND     R4,AUX
        NZT     AUX,*+2
        JMP     SHIFT
```

Figure 6

```
        MOVE    OVF, AUX
        ADD     R5,AUX
        ADD     R1,R1
        SHIFT THE PARTIAL PRODUCT AND THE
        MULTIPLIER ONE PLACE TO THE RIGHT.
SHIFT   XMIT    177H,AUX
        AND     R4(1),R4
        XMIT    200H,AUX
        AND     R3(1),AUX
        XOR     R4,R4
        XMIT    177H,AUX
        AND     R3(1),R3
        XMIT    200H,AUX
        AND     R2(1),AUX
        XOR     R3,R3
        XMIT    177H,AUX
        AND     R2(1),R2
        XMIT    200H,AUX
        AND     R1(1),AUX
        XOR     R2,R2
        XMIT    177H,AUX
        AND     R1(1),R1
        SHIFTED 16 TIMES YET?
        XMIT    1,AUX
        ADD     R11,R11
        NZT     R11,LOOP
        NEGATE RESULT IF NECESSARY.
        ORG     14,32
        NZT     SRESULT,ENDD
        XMIT    377H,AUX
        XOR     R4,R4
        XOR     R3,R3
        XOR     R2,R2
        XOR     R1,R1
        XMIT    1,AUX
        ADD     R4,R4
        MOVE    OVF,AUX
        ADD     R3,R3
        MOVE    OVF,AUX
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        SEL     LINK
        MOVE    LINK,R11
        RTN
        END M16X16
```

Figure 6 (Cont'd)

PROCEDURE NAME: DIV16X8

General Description:

DIV16X8 is a procedure for doing integer division with binary numbers in 2's complement notation. To improve execution speed, negative numbers are converted to positive, but the result is given the proper sign at the end of the computation. This procedure divides the contents of R1, R2 by the contents of R3. The quotient is placed in R2 and the remainder is placed in R1. Note that this implies an accuracy of 8 bits. If the quotient is greater than 8 bits, only the high order byte will be generated, and the remainder will be greater than the divisor. See Figure 7 for the flow chart and Figure 8 for the program listing.

Memory Requirements:

Program Storage: 59 words
Working Storage: 1 byte

Registers Used And Their Logical Function:

R1 This register originally holds the high order byte of the dividend. When the procedure is completed, R1 contains the remainder.

R2 This register originally holds the low order byte of the dividend. When the procedure is completed, R2 contains the quotient.

R3 This register originally holds the divisor.

R4 This register is used as a counter to determine when the division is completed.

R5 This register contains information used to compute the sign of the result.

R6 This register holds the 2's complement of the divisor.

R11 This register is used as temporary storage.

Timing:

Average if both operands are negative: 37.5μs

Average if both operands are positive: 35.5μs

Worst Case: 39.75μs

Calls On Other Library Procedures: None

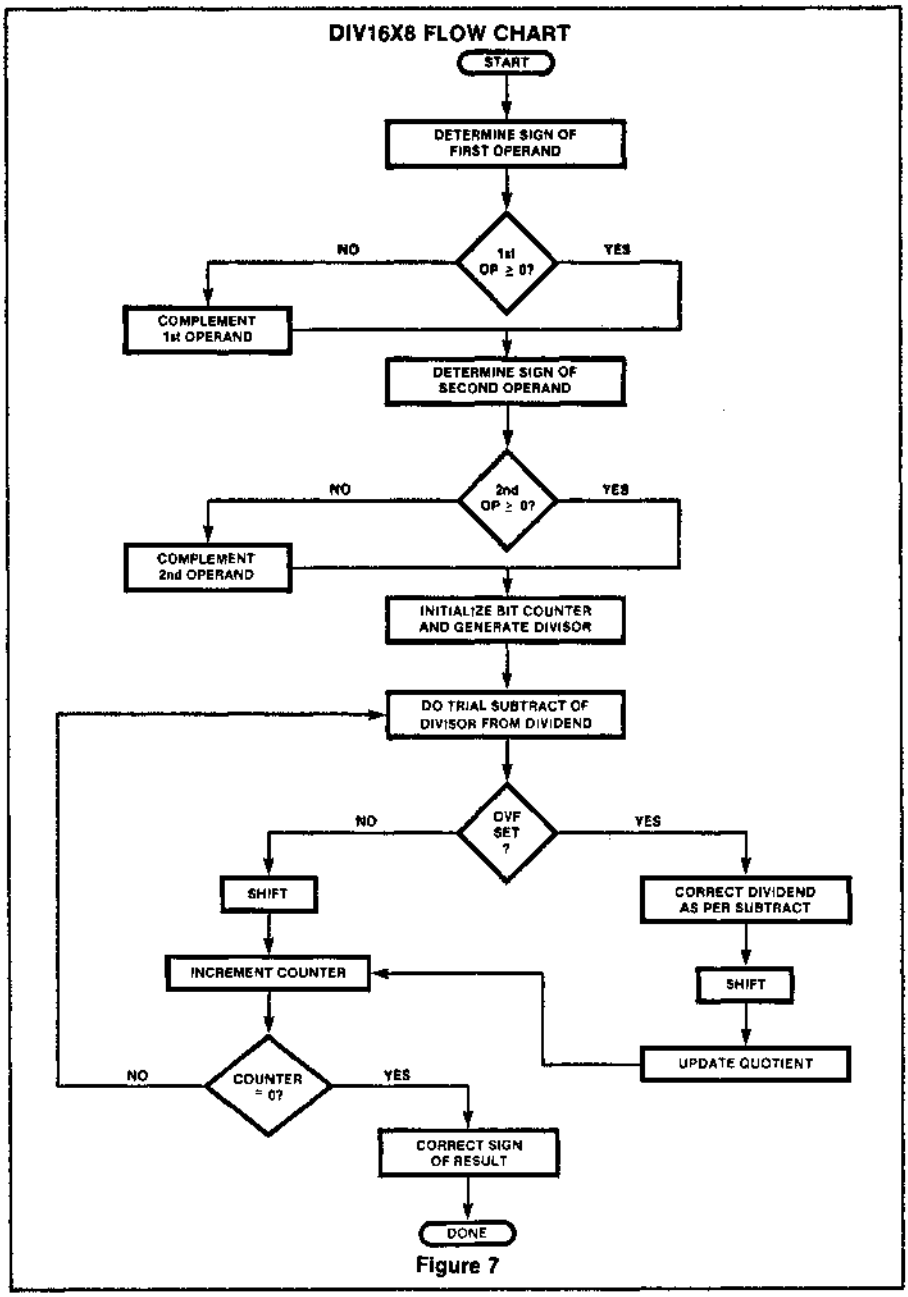


Figure 7

DIV16X8 PROGRAM LISTING

```
        PROC DIV16X8
        16-BIT BY 8-BIT 2's COMP DIVIDE WITH DIVIDEND
        IN R1,R2 AND DIVISOR IN R3
LINK RIV 200,7,8
        SEL   LINK          SAVE LINK.
        MOVE  R11,LINK
OP1     XMIT  1,AUX         ACCESS/TEST OP11 SIGN
        AND   R1(7),AUX
        XEC   TAB(AUX)
        NZT   AUX,COMP1
        JMP   OP2
TAB     XMIT  1,R5          POSITIVE
        XMIT  0,R5
COMP1   XMIT  377H,AUX      2's COMP DIVIDEND
        XOR   R1,R1
        XOR   R2,R2
        XMIT  1,AUX
        ADD   R2,R2
        MOVE  OVF,AUX
        ADD   R1,R1
OP2     XMIT  1,AUX
        AND   R3(7),AUX     ACCESS/TEST DIVISOR SIGN
        XOR   R5,R5
        NZT   AUX,COMP2
        JMP   START
COMP2   XMIT  377H,AUX
        XOR   R3,R3         2's COMPLEMENT DIVISOR
        XMIT  1,AUX
        ADD   R3,R3
START   XMIT  -9,R4
        XMIT  377H,AUX      NEGATIVE OF DIVISOR
        XOR   R3,R6
        XMIT  1,AUX
        ADD   R6,R6
        ORG   20,256
LOOP    MOVE  R6,AUX
        ADD   R1,R11        TRIAL SUBTRACT
        ORG   8,256
        NZT   OVF,DIV1
DIV0    MOVE  R2,AUX        SHIFT LEFT 1
        ADD   R2,R2
        MOVE  OVF,AUX
        ADD   R1,AUX
        ADD   R1,R1
        JMP   CHECK
DIV1    MOVE  R11,R1        UPDATE DIVIDEND
        MOVE  R2,AUX
        ADD   R2,R2
        MOVE  OVF,AUX
        ADD   R1,AUX
        ADD   R1,R1
        XMIT  1,AUX         SHIFT ON 1
        XOR   R2,R2
        NZT   R4,LOOP       DONE?
        ORG   9,256
        NZT   R5,ENDD       YES. TEST SIGN RESULT
        XMIT  377H,AUX
        XOR   R1,R1
        XOR   R2,R2
        XMIT  1,AUX
        ADD   R1,R1
        MOVE  OVF,AUX
        ADD   R2,R2
ENDD    SEL   LINK          RESTORE RETURN LINK.
        MOVE  LINK,R11
        RTN
        END   DIV16X8
```

Figure 8

PROCEDURE NAME: D24X12

General Description:

D24X12 is a procedure to do integer division with 2's complement numbers. To improve execution speed, any negative operands are first converted to positive. However, the quotient is given the proper sign at the end of the computation. D24X12 divides the contents of R1, R2, R3 by the 4 low order bits of R5 and R4. The quotient is stored in the 4 low order bits of R2 and R3. The remainder is stored in R1 and the 4 high order bits of R2. Note this implies a 12-bit result. If the quotient is greater than 12 bits only the 12 most significant bits will be generated, and the remainder may be larger than the divisor. Also note that if the divisor supplied by the main program is zero, then the dividend is merely rotated 12 places. See Figure 9 for the flow chart and Figure 10 for the program listing.

Memory Requirements:

Program Storage: 94 words
Working Storage: 2 bytes

Registers Used And Their Logical Function:

R1 This register originally contains the high order byte of the dividend. At the end of the procedure, it contains the 8 high order bits of the remainder.

R2 This register originally contains the middle byte of the dividend. At the end of the procedure, it contains the 4 low order bits of the remainder and the 4 high order bits of the quotient.

R3 This register originally contains the low order byte of the dividend. At the end of the procedure, it contains the 8 low order bits of the quotient.

R4 This register originally contains the 4 high order bits of the divisor. Shortly after the procedure begins execution, R4 contains the 8 high order bits of the divisor.

R5 This register originally contains the 8 low order bits of the divisor. Shortly after the procedure begins execution, R5 contains the 4 low order bits of the divisor left justified.

R6 This register contains the high order 8 bits of the negated divisor.

R11 This register contains the 4 low order bits of the negated divisor. These bits are left justified.

Timing:

Worst Case: 109.5μs

Calls On Other Library Procedures: None

D24X12 FLOW CHART

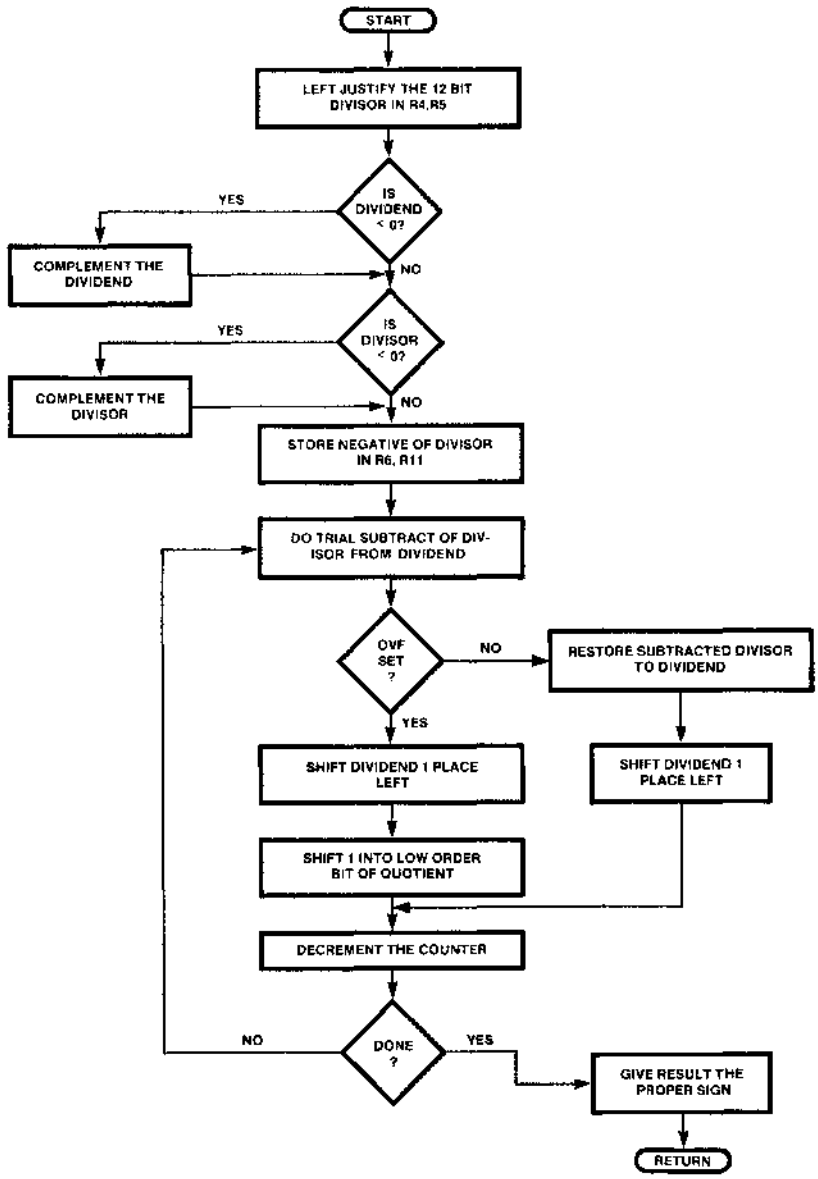


Figure 9

D24X12 PROGRAM LISTING

```
        PROC D24X12

        MCLIB PROCEDURE TO DIVIDE A 24-BIT 2's
        COMPLEMENT NUMBER. BY A 16-BIT 2's
        COMPLEMENT NUMBER.

LINK RIV 200,7,8
TEMP RIV 201,7,8
TEMPH RIV 201,3,4
SIGN RIV 201,7,1
COUNT RIV 201,6,5
SWITCH RIV 201,0,1
        SEL     LINK
        MOVE    R11,LINK        SAVE RETURN LINK.
BEGIN   XMIT    360H,AUX        ALIGN DIVISOR BY
        SEL     TEMP            SHIFTING LEFT FOUR
        MOVE    R5,TEMP         PLACES.
        AND     R5(4),R5
        AND     R4(4),AUX
        ADD     TEMPH,R4
        XMIT    -13,COUNT
OP1     XMIT    1,AUX
        AND     R1(7),AUX
        XEC     TAB(AUX)
        NZT     AUX,COMP1
        JMP     OP2
TAB     XMIT    1,SIGN
        XMIT    0,SIGN
COMP1   XMIT    377H,AUX
        XOR     R3,R3
        XOR     R2,R2
        XOR     R1,R1
        XMIT    1,AUX
        ADD     R3,R3
        MOVE    OVF,AUX
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        XMIT    1,AUX
OP2     AND     R4(7),AUX
        XOR     SIGN,SIGN
        NZT     AUX,COMP2
        JMP     START
COMP2   XMIT    377H,AUX
        XOR     R4,R4
        XMIT    360H,AUX
        XOR     R5,R5
        XMIT    20H,AUX
        ADD     R5,R5
        MOVE    OVF,AUX
        ADD     R4,R4
START   XMIT    377H,AUX        NEGATE DIVISOR.
        XOR     R4,R6
        XMIT    360H,AUX
        XOR     R5,R11
        XMIT    20H,AUX
        ADD     R11,R11
        MOVE    OVF,AUX
        ADD     R6,R6
        ORG     32
```

Figure 10

```
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        MOVE    R6,AUX
        ADD     R1,R1
        MOVE    OVF,SWITCH
        NZT     OVF,DIV1
DIV0    MOVE    R5,AUX          RESTORE ADD.
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        MOVE    R4,AUX
        ADD     R1,R1
DIV1    MOVE    R3,AUX          SHIFT LEFT 1 PLACE.
        ADD     R3,R3
        MOVE    OVF,AUX
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        MOVE    R2,AUX
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        MOVE    R1,AUX
        ADD     R1,R1
        XMIT    1,AUX
        NZT     SWITCH,*+2
        JMP     CHECK
        XOR     R3,R3           BUILD UP QUOTIENT.
CHECK   ADD     COUNT,COUNT
        NZT     COUNT,LOOP
        ORG     15,32
        NZT     SIGN,ENDD
        XMIT    377H,AUX        GIVE RESULT THE
        XOR     R1,R1           PROPER SIGN.
        XOR     R2,R2
        XOR     R3,R3
        XMIT    20H,AUX
        ADD     R2,R2
        MOVE    OVF,AUX
        ADD     R1,R1
        MOVE    OVF,AUX
        ADD     R3,R3
        MOVE    OVF,AUX
        ADD     R2,R2
ENDD    SEL     LINK            RESTORE RETURN LINK.
        MOVE    LINK,R11
        RTN
        END     D24X12
```

Figure 10 (Cont'd)

PROCEDURE NAME: DTOB

General Description:

DTOB is a procedure which converts 7 unsigned BCD digits, stored in consecutive Working Storage locations, into up to 24 bits of binary which is stored in R2, R3, R4. The main program must supply in R6 the byte address of the first (low order) BCD digit. DTOB expects to find this first digit right justified in the Working Storage byte. The first 6 BCD digits should be packed 2 per byte in ascending memory locations. The seventh BCD digit should be right justified in the last memory location. DTOB accomplishes the conversion by adding the proper power of 10 the proper number of times to R2, R3, R4. Finally, since most applications where BCD to binary conversion is necessary are not expected to require high speed, DTOB had traded off some speed in order to limit program storage requirements. See Figure 11 for the flow chart and Figure 12 for the program listing.

Memory Requirements:

Program Storage: 56 words
Working Storage: 4½ bytes

Registers Used And Their Logical Functions:

- R1 This is a program control register used to step through Working Storage and to properly access either the high or low order four bits of an enabled Working Storage location.
- R2 This register contains the high 8 bits of the binary result.
- R3 This register contains the middle 8 bits of the binary result.
- R4 This register contains the low 8 bits of the binary result.
- R5 This register is loaded with the current BCD digit and is then used as a counter to add a power of 10 the proper number of times.
- R6 This register is used to step through Working Storage. It originally contains the address of the first BCD digit to be converted, which is supplied by the main program.
- R11 The contents of this register are used to select the proper power of 10 to be added to the partial binary sum. It is also used to terminate the procedure.

Timing:

Worst Case: 222.75μs (conversion of 9,999,999 to binary)
Average Case: 132.75μs (conversion of 5,555,555 to binary)

Calls On Other Library Procedures: None

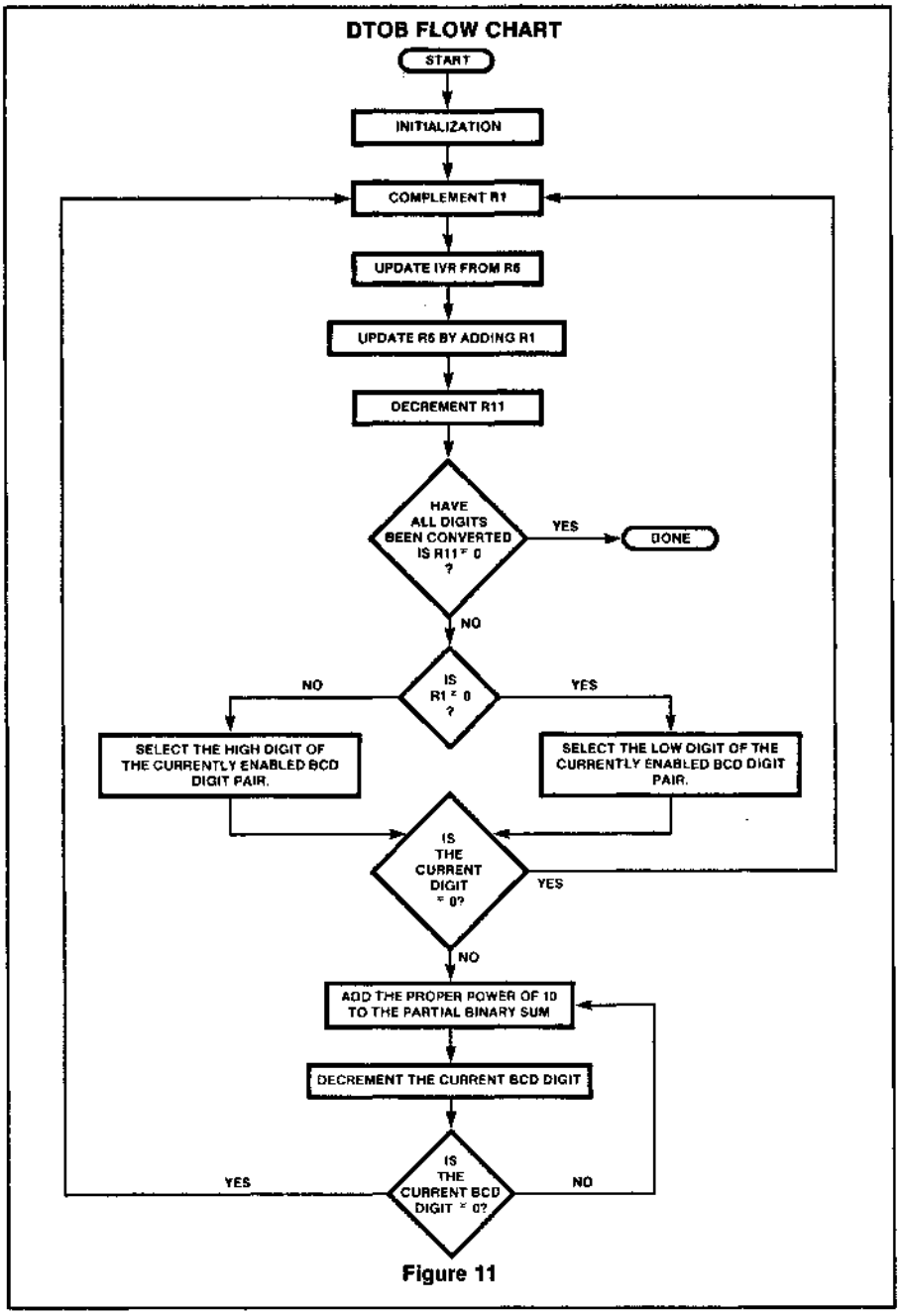


Figure 11

DTOB PROGRAM LISTING

```
        PROC DTOB

        MCLIB PROCEDURE TO CONVERT 7 BCD
        DIGITS INTO UPTO 24 BITS OF BINARY.

LINK RIV 200,7,8
DMYL RIV 200,7,4
DMYH RIV 200,3,4

        SEL     LINK        SAVE RETURN LINK.
        MOVE    R11,LINK
START   XMIT    0,R3        INITIALIZATION.
        XMIT    0,R2
        XMIT    0,R1
        XMIT    7,R11
        MOVE    R6,IVR
ONE     MOVE    DMYL,R4
        ORG     25,256
LOOP1   XEC     TABA(R1)    COMPLEMENT R1.
        MOVE    R1,AUX
        MOVE    R6,IVR
        ADD     R6,R5       INCREMENT ONLY WHEN
        XMIT    -1,AUX      NECESSARY.
        ADD     R11,R11     DECREMENT COUNTER.
        NZT     R11,*+2
        JMP     DONE
        XEC     TAB1(R1)
        NZT     R5,LOOP2
        JMP     LOOP1
        ORG     25,256
LOOP2   XEC     TAB3-1(R11)
        ADD     R4,R4
        ORG     29,256
        XEC     TAB4-1(R11)
        ADD     OVF,AUX
        ADD     R3,R3
        ORG     32,256
        XEC     TAB5-1(R11)
        ADD     OVF,AUX
        ADD     R2,R2
        XMIT    -1,AUX
        ADD     R5,R5
        NZT     R5,LOOP2
        NZT     R11,LOOP1
DONE    SEL     LINK        RESTORE RETURN LINK..
        MOVE    LINK,R11
        RTN
TABA    XMIT    1,R1
        XMIT    0,R1
TAB1    MOVE    DMYL,R5
        MOVE    DMYH,R5
TAB3    XMIT    100H,AUX    MILLION.
        XMIT    240H,AUX    HUNDRED THOUSAND.
        XMIT    20H,AUX     TEN THOUSAND.
        XMIT    350H,AUX    THOUSAND.
        XMIT    144H,AUX    HUNDRED
        XMIT    12H,AUX     TEN.
TAB4    XMIT    102H,AUX    MILLION.
        XMIT    206H,AUX    HUNDRED THOUSAND.
        XMIT    47H,AUX     TEN THOUSAND.
        XMIT    3,AUX       THOUSAND.
        XMIT    0,AUX       HUNDRED.
        XMIT    0,AUX       TEN.
TAB5    XMIT    17H,AUX     MILLION.
        XMIT    1,AUX       HUNDRED THOUSAND.
        XMIT    0,AUX       TEN THOUSAND.
        XMIT    0,AUX       THOUSAND.
        XMIT    0,AUX       HUNDRED.
        XMIT    0,AUX       TEN.
        END DTOB
```

Figure 12

PROCEDURE NAME: BTOD

General Description:

BTOD is a procedure which converts a 24-bit positive binary number into 7 BCD digits which are stored in ascending order in consecutive Working Storage locations. The contents of registers R2, R3, R4 are taken to be the binary number with R2 being the high order byte. The main program must supply in R6 the absolute address of the Working Storage byte in which the two low order BCD digits are to be stored. BTOD accomplishes the conversion by keeping track of the number of times it can subtract decreasing powers of 10 from the binary number. See Figure 13 for the flow chart and Figure 14 for the program listing.

Memory Requirements:

Program Storage: 87 words
Working Storage: $4\frac{1}{2}$ bytes

Registers Used and Their Logical Functions:

R1 This is a program control register used to step through Working Storage and to properly access either the high or the low order BCD digit of an enabled Working Storage byte.

R2 This register originally contains the high order byte of the binary number. However, note that the binary number is destroyed as the BCD digits are computed.

R3 This register originally contains the middle byte of the binary number.

R4 This register originally contains the low order byte of the binary number.

R5 The contents of this register are used to select the proper power of 10 to be subtracted from the partial binary difference. It is also used to terminate the procedure.

R6 This register is used to step through Working Storage. It originally contains the address of the storage location for the first BCD digit pair.

R11 This register is used as temporary storage of overflow bits.

Timing:

Worst Case: 346.67μs (Conversion of 46114163_8 to BCD)

Calls On Other Library Procedures: None

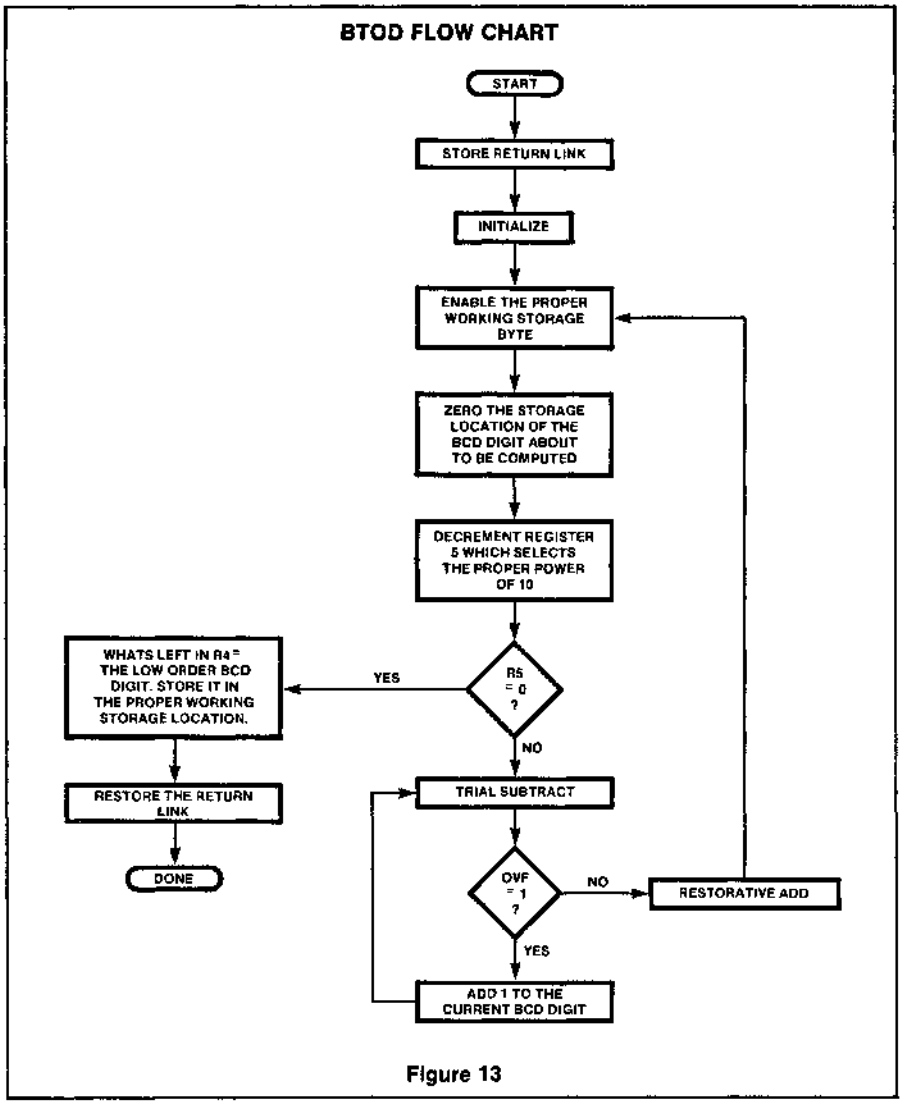


Figure 13

BTOD PROGRAM LISTING

```
        MCLIB PROCEDURE TO CONVERT A 24-BIT
        POSITIVE BINARY NUMBER INTO 7 BCD DIGITS
LINK RIV 200,7,8
DMYL RIV 200,7,4
DMYH RIV 200,3,4
        SEL   LINK          STORE RETURN LINK.
        MOVE  R11,LINK
START   XMIT  7,R5
        XMIT  0,R1
        XMIT  3,AUX         ALIGN POINTER TO HIGH
        ADD   R6,R6         ORDER BCD DIGIT.
        ORG   41,256
LOOP1   XEC   TABA+1(R1)    FLIP R1
        MOVE  R1,AUX
        MOVE  R6,IVR        ENABLE PROPER W.S. BYTE
        ADD   R6,R6         DECREMENT R6 WHEN NEC.
        ORG   39,256
        XEC   TABC+1(R1)    ZERO BCD DIGIT LOC.
        XMIT  -1,AUX
        ADD   R5,R5
        NZT   R5,*+2
        JMP   DONE
        ORG   42,256
LOOP2   XEC   TAB3-1(R5)
        ADD   R4,R4
        ORG   46,256
        XEC   TAB4-1(R5)
        ADD   OVF,AUX
        MOVE  OVF,R11
        ADD   R3,R3
        MOVE  OVF,AUX
        ADD   R11,R11
        ORG   46,256
        XEC   TAB5-1(R5)
        ADD   R11,AUX
        MOVE  OVF,R11
        ADD   R2,R2,
        NZT   OVF,OK
        NZT   R11,OK
        ORG   46,256
RESTORE XEC   TAB6-1(R5)
        ADD   R4,R4
        ORG   50,256
        XEC   TAB7-1(R5)
        ADD   OVF,AUX
        ADD   R3,R3
        ORG   53,256
        XEC   TAB8-1(R5)
        ADD   OVF,AUX
        ADD   R2,R2
        JMP   LOOP1
OK      XMIT  1,AUX
        XEC   TABB+1(R1)
        JMP   LOOP2
DONE    MOVE  R4,DMYL
        SEL   LINK          WHAT WAS LEFT IN R4 WAS
        MOVE  LINK,R11      RESTORE LINK.
        RTN
```

Figure 14

```
        XMIT  -1,R1
TABB    ADD   DMYL,DMYL
        ADD   DMYH,DMYH
TABC    XMIT  0,DMYL
        XMIT  0,DYMH
        SUBTRACTION TABLES.
TAB3    XMIT  366H,AUX      TEN.
        XMIT  234H,AUX      HUNDRED.
        XMIT  30H,AUX       THOUSAND.
        XMIT  360H,AUX      TEN THOUSAND.
        XMIT  140H,AUX      HUNDRED THOUSAND.
        XMIT  300H,AUX      MILLION.
TAB4    XMIT  377H,AUX      TEN.
        XMIT  377H,AUX      HUNDRED.
        XMIT  374H,AUX      THOUSAND.
        XMIT  330H,AUX      TEN THOUSAND.
        XMIT  171H,AUX      HUNDRED THOUSAND.
        XMIT  275H,AUX      MILLION.
TAB5    XMIT  377H,AUX      TEN.
        XMIT  377H,AUX      HUNDRED.
        XMIT  377H,AUX      THOUSAND.
        XMIT  377H,AUX      TEN THOUSAND.
        XMIT  376H,AUX      HUNDRED THOUSAND.
        XMIT  360H,AUX      MILLION.
        ADDITION TABLES.
TAB6    XMIT  12H,AUX       TEN.
        XMIT  144H,AUX      HUNDRED.
        XMIT  350H,AUX      THOUSAND.
        XMIT  20H,AUX       TEN THOUSAND.
        XMIT  240H,AUX      HUNDRED THOUSAND.
        XMIT  100H,AUX      MILLION.
TAB7    XMIT  0,AUX         TEN.
        XMIT  0,AUX         HUNDRED.
        XMIT  3,AUX         THOUSAND.
        XMIT  47H,AUX       TEN THOUSAND.
        XMIT  206H,AUX      HUNDRED THOUSAND.
        XMIT  102H,AUX      MILLION.
TAB8    XMIT  0,AUX         TEN.
        XMIT  0,AUX         HUNDRED.
        XMIT  0,AUX         THOUSAND.
        XMIT  0,AUX         TEN THOUSAND.
        XMIT  1,AUX         HUNDRED THOUSAND.
        XMIT  17H,AUX       MILLION.
        END BTOD
```

Figure 14 (Cont'd.)

PROCEDURE NAME: SORT

General Description:

SORT is a procedure which sorts the contents of a block of Working Storage locations into descending order. That is, the data is sorted so that as the Working Storage addresses increase, the value of the contents decrease. The boundaries of the block are set by the main program. The lower address boundary must be placed in R1. The high address boundary must be placed in R2. The block must be contained within a single memory page and that page must be selected by the main program. The contents of the block are treated as 2's complement numbers. See Figure 15 for the flow chart and Figure 16 for the program listing.

Memory Requirements:

Program Storage: 47 words
Working Storage: None

Registers And Their Logical Function:

- R1 This register is used to pass the lower boundary address to the procedure. In the course of execution, this value is changed to its 2's complement.
- R2 This register is used to pass the upper boundary address to the procedure. In the course of execution, this value is changed to its 2's complement.
- R3 This register is used to hold the current address, N.
- R4 This register is used to hold the current contents of N which is denoted as (N).
- R6 This register is used as a scratchpad to hold a variety of temporary results.

Timing:

It is difficult to compute the exact timing for this procedure. Six microseconds per byte sorted is a realistic average time.

Calls On Other Library Procedures: None

SORT FLOW CHART

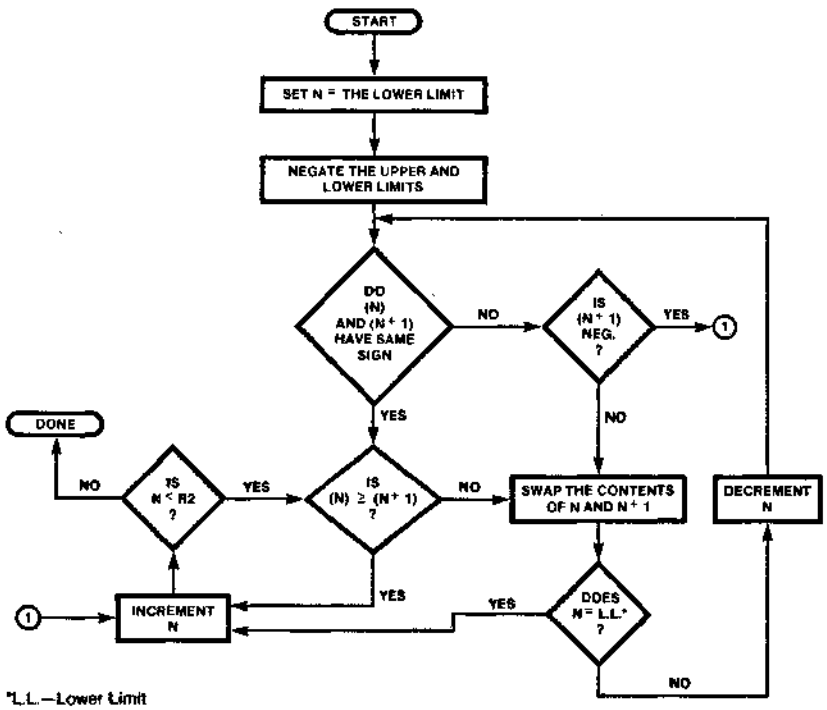


Figure 15

SORT PROGRAM LISTING

```
          PROC SORT
DMY RIV 200,7,6
SIGN RIV 200,0,1
START     MOVE   R1,R3        R3 = N.
          XMIT   -1,AUX       NEGATE R2
          ADD    R2,R2
          XOR    R2,R2
          ADD    R1,R1        NEGATE R1.
          XOR    R1,R1

          MOVE   R3,IVR       ENABLE N.
          MOVE   DMY,R4       R4 = (N).

          ORG    21,255
TEST      XMIT   1,AUX
          ADD    R3,IVR       ENABLE N + 1.

          AND    R4(7),R6     R6 = SIGN OF (N).
          MOVE   SIGN,AUX     AUX = SIGN OF (N + 1).
          XOR    R6,R6        R6 = 1 IF DIFF.
          ORG    18,256
          NZT    R6,CHECK

          XMIT   -1,AUX
          ADD    DMY,R6
          XOR    R6,AUX       -(N + 1) NOW IN AUX.
          ADD    R4,R6        (N) - (N + 1) NOW IN R6.
          XMIT   1,AUX
          AND    R6(7),R6     R6 = 0,IF (N) = (N + 1).
          ORG    13,256
          NZT    R6,SWAP

          ORG    10,256
NEXTN     ADD    R3,R3        INCREMENT N.
          MOVE   R3,IVR
          MOVE   DMY,R4

          MOVE   R2,AUX       AUX = -UPPER LIMIT.
          ADD    R3,R6        R6 = N - R2
          XMIT   1,AUX
          AND    R6(7),R6     N - R2 MUST BE < = 0.
          NZT    R6,TEST      R6 = 0 IF N = R2.
          JMP    DONE

CHECK     NZT    AUX,NEXTN    AUX = SIGN OF (N + 1).
          JMP    SWAP

SWAP      MOVE   DMY,R6       R6 = (N + 1).
          MOVE   R4,DMY
          MOVE   R3,IVR       ENABLE N.
          MOVE   R6,DMY

          MOVE   R1,AUX
          ADD    R3,R6
          NZT    R6,DECN
          XMIT   1,AUX
          JMP    NEXTN
DECN      XMIT   -1,AUX
          ADD    R3,R3
          MOVE   R3,IVR
          MOVE   DMY,R4
          JMP    TEST

DONE      RTN
          END SORT
```

Figure 16

PROCEDURE NAME: FPADD

General Description:

FPADD performs a double precision floating point addition of two operands located anywhere in the same page of Working Storage. These floating point operands must have an 8-bit exponent and a 16-bit mantisa. If the exponent of an operand is found in memory location N, then the high order byte of the mantisa must be in location N + 1. The low order byte must be in location N + 2. Both the exponent and the mantisa must be represented in 2's complement notation. The calling program specifies the operands to be added by passing the addresses of their exponents in R1 and R2 and by selecting the proper memory page before calling FPADD.

FPADD begins processing by comparing the relative sizes of the two exponents. Because the precision of the addition is 16 bits, if the exponents differ by more than 15, the answer will be equal to the larger of the two operands. If the addition is meaningful, FPADD computes the exponent of the result and then calls a procedure to adjust the smaller of the operands so that the exponents of the operands are equal. FPADD then calls another procedure which performs the actual 16-bit addition. See Figure 17 for the flow chart and Figure 18 for the program listing.

Memory Requirements:

Program Storage: 38 words
Working Storage: 1 byte

Registers Used And Their Logical Funtions:

R1 This register contains the address of the exponent of the first operand. This address must be supplied by the main program.

R2 This register contains the address of the exponent of the second operand. This address must be supplied by the main program. R2 is also used to return the address of the exponent of the answer to the main program.

R3 Initially this register is used to hold the value of the exponent of the first operant (X1). Later it is used to pass the address of the exponent of the operand which will be adjusted before the addition takes place.

R5 This register is used as a scratchpad throughout most of the procedure, but is also used to pass the address of the high order byte of the first operand to the addition procedure.

R6 This register is used as a scratchpad, but also passes the number of places which the smaller of the two operands must be shifted. Finally, it is used to pass the address of the high order byte of the second operand to the addition procedure.

R11 In programs written in MCCAP, this register is used to hold the return link for subroutines.

Timing:

Worst Case (exponents differ but by less than 16, and the addition overflows)
≈ 22 microseconds. This is the time from when FPADD is called to when it returns.

Calls On Other Library Procedures:

DSHIFT
TAD16F

FPADD FLOW CHART

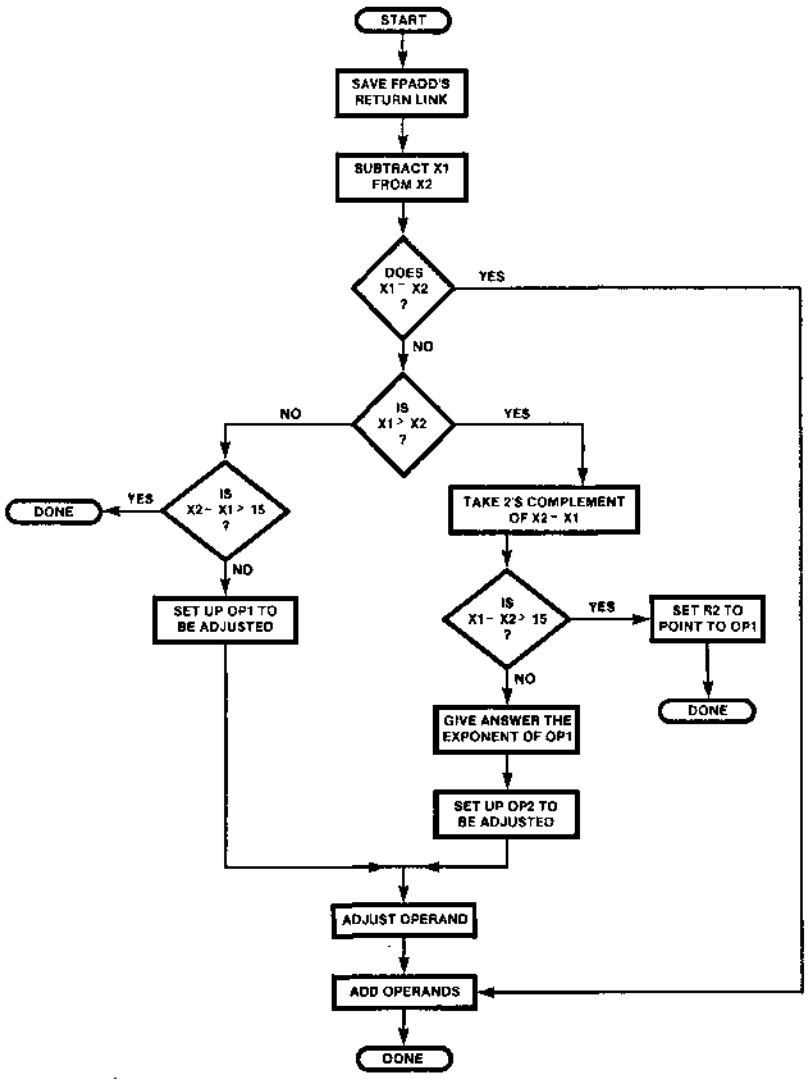


Figure 17

FPADD PROGRAM LISTING

```
        PROC FPADD

        MCLIB PROCEDURE TO ADD TWO FLOATING POINT
        DOUBLE PRECISION 2's COMPLEMENT NUMBERS.

LINK RIV 200,7,8
X1 RIV 200,7,8
X2 RIV 200,7,8

NOTE THAT X1 AND X2 ARE NEVER TRANSMITED TO
THE IVR REG. THUS THE ADDRESSES ASIGNED TO
THESE VARIABLES ARE NOT IMPORTANT.
        SEL   LINK      SAVE FPADD RETURN LINK.
        MOVE  R11,LINK
START   MOVE  R1,IVR    ENABLE X1.
        MOVE  X1,R3
        MOVE  R2,IVR    ENABLE X2.
        XMIT  -1,AUX
        ADD   R3,R6     TAKE 2's COMP OF X1.
        XOR   R6,AUX    -X1 IN AUX.
        ADD   X2,R6     X2-X1 IN R6.
        NZT   R6,*+2
        JMP   ADDD      EXPONENTS MATCH.
        XMIT  1,AUX
        AND   R6(7),R4  R5= SIGN OF X2 -X1.
        NZT   R5,NEG
POS     XMIT  -16,AUX   R6 IS POS. MEANING X2 IS
                        GREATER THAN X1.
        ADD   R6,R5     DO X2 AND X1 DIFFER BY
                        MORE THAN 15?
        NZT   OVF,DONE  OP2 IS THE ANSWER.
        MOVE  R1,R3     THE ADDITION IS POSSIBLE BUT
        JMP   ADJ       OP1 MUST BE SHIFTED.
NEG     XMIT  -1,AUX    R6 IS NEG, MEANING X1 IS
                        GREATER THAT X2.
        ADD   R6,R6     MAKE R6 POSITIVE.
        XOR   R6,R6
        XMIT  -15,AUX   DO X1 AND X2 DIFFER BY MORE
                        THAN 15?
        ADD   R5,R5
        NZT   OVF,SWAP  OP1 IS THE ANSWER.
        MOVE  R3,X2     THE ANSWER WILL BE IN OP2
                        AND HAVE X1 AS AN EXPONENT.
        MOVE  R2,R3     THE ADDITION IS POSSIBLE BUT
        JMP   ADJ       OP2 MUST BE SHIFTED.
SWAP    MOVE  R1,R2     WHEN THE PROC RETURNS,R2
        JMP   DONE      MUST HOLD THE ADDRESS OF THE
                        RESULT.
ADJ     CALL  DSHIFT
ADDD    XMIT  1,AUX
        ADD   R1,R5     R5 POINTS TO HOB OF OP1.
        ADD   R2,R6     R6 POINTS TO HOB OF OP2.
        CALL  TAD16F
DONE    SEL   LINK      RESTORE FPADD RETURN LINK.
        MOVE  LINK,R11
        RTN             AT THIS POINT R2 POINTS TO
        END FPADD       THE ANSWER.
```

Figure 18

PROCEDURE NAME: DSHIFT

General Description:

DSHIFT is a program storage intensive high speed procedure which shifts a double precision variable on the right bank IV 1 to 15 bits to the right. The execution time is independent of the number of bits to be shifted and is fixed. Changing the 3 occurences of "IVR" in the procedure to "IVL" converts DSHIFT to operate on left bank IV variables. If the high order byte of the double precision variable is in location N, then the low order byte must be in location N + 1. See Figure 19 for the flow chart and Figure 20 for the program listing.

Memory Requirements:

Program Storage: 52 words
Working Storage: None

Registers Used And Their Logical Functions:

R3 This register is used to pass the address of the variable's high order byte to the procedure.
R4 This register is used to hold and rotate the variable's high order byte.
R5 This register is used to hold and rotate the variable's low order byte.
R6 This register is used to pass the number of places that the variable must be shifted to the procedure. If the contents of R6 is outside the range of 1 to 15 errors will occur.
R11 In programs written for MCCAP, this register holds the procedure return link.

Timing:

4.75 microseconds

Calls On Other Library Procedures: None

DSHIFT PROGRAM LISTING

```
        PROC DSHIFT

        MCLIB PROCEDURE TO SHIFT A DOUBLE
        PRECISION VARIABLE IN WORKING STORAGE
        1 TO 15 PLACES TO THE RIGHT.

DMY RIV 201,7,8
H1 RIV 201,0,1
H2 RIV 201,1,2
H3 RIV 201,2,3
H4 RIV 201,3,4
H5 RIV 201,4,5
H6 RIV 201,5,6
H7 RIV 201,6,7

L1 RIV 201,7,1
L2 RIV 201,7,2
L3 RIV 201,7,3
L4 RIV 201,7,4
L5 RIV 201,7,5
L6 RIV 201,7,6
L7 RIV 201,7,7

NOTE THAT NONE OF THE VARIABLES DECLARED IN THIS
PROCEDURE ARE EVER TRANSMITTED TO THE IVR REG.
THUS IT IS NOT IMPORTANT WHAT ADDRESS THEY ARE
ASIGNED TO.

SHIFT   XMIT   1,AUX
        ADD    R3,IVR      ENABLE HIGH ORDER BYTE.
        MOVE   DMY,R4
        XMIT   0,DMY
        XMIT   2,AUX
        ADD    R3,IVR      ENABLE LOW ORDER BYTE.
        MOVE   DMY,R5
        ORG    40,256
        XEC    TAB1-1(R6)
        MOVE   R5,DMY
        XEC    TAB2-1(R6)
        ORG    45,256
        XEC    TAB3-1(R6)
        XMIT   1,AUX
        ADD    R3,IVR      ENABLE HIGH ORDER BYTE.
        XEC    TAB4-1(R5)
DONE    RTN
```

Figure 20

```
TAB2    MOVE   R4,H1
        MOVE   R4,H2
        MOVE   R4,H3
        MOVE   R4,H5
        MOVE   R4,H6
        MOVE   R4,H7
        MOVE   R4,DMY
TAB4    MOVE   R4,L7
        MOVE   R4,L6
        MOVE   R4,L5
        MOVE   R3,L4
        MOVE   R4,L3
        MOVE   R4,L2
        MOVE   R4,L1
TAB1    MOVE   R5(1),R5
        MOVE   R5(2),R5
        MOVE   R5(3),R5
        MOVE   R5(4),R5
        MOVE   R5(5),R5
        MOVE   R5(6),R5
        MOVE   R5(7),R5
        NOP
TAB3    MOVE   R4(1),R4
        MOVE   R4(2),R4
        MOVE   R4(3),R4
        MOVE   R4(4),R4
        MOVE   R4(5),R4
        MOVE   R4(6),R4
        MOVE   R4(7),R4
        NOP
        XMIT   0,H1
        XMIT   0,H2
        XMIT   0,H3
        XMIT   0,H4
        XMIT   0,H5
        XMIT   0,H6
        XMIT   0,H7
        END DSHIFT
```

Figure 20 (Cont'd.)

PROCEDURE NAME: TAD16F

General Description:

TAD16F is a modified version of TAD16 designed to be used as part of a floating point adder. TAD16F accepts two double precision operands located anywhere on the same Working Storage memory page. The program which calls TAD16F must specify these operands by selecting the proper memory page and by passing the addresses of the high order bytes of the operands to the procedure in registers.

If the double precision add overflows, the procedure increments the exponent whose address must be supplied in R2 and then shifts the 16-bit sum one place to the right. There is no warning given if the exponent overflows, but this is a very unlikely occurrence. See Figure 21 for the flow chart and Figure 22 for the program listing.

WARNING! In the event of an overflow, TAD16F shifts the 16-bit sum in Working Storage by taking advantage of the wrap-around read characteristics of the MicroController's Working Storage locations. In practice, if you try to read a Working Storage data field which is so long it runs off the end of the byte, the MicroController fills in the high order bits by reading the unused low order bits of the byte. For example, if we try to read a field 8 bits long and whose right most bit is bit 6, then bit 7 will be used as the high order bit of the field.

This is a perfectly valid MicroController practice. MCCAP will generate an error flag, but assemble the instruction anyway. In TAD16F, there are two occurrences of the instruction MOVE A,DMY. These instructions should assemble as 0 36037.

Memory Requirements:

Program Storage: 35 words
Working Storage: None

Registers Used And Their Logical Functions:

R2 This register contains the address of the sum's exponent. It must be supplied by the calling program.

R3 This register is used to hold the computed address of the low order byte of the second operand.

R4 This register is used to hold computed information on the signs of the two operands.

R5 This register is used to pass the address of the high order byte of the first operand to TAD16F.

R6 This register is used to pass the address of the high order byte of the second operand to TAD16F.

R11 For programs written in MCCAP, this register is used to hold TAD16F's return link.

Timing:

Worst Case: 7.75 microseconds

Calls On Other Library Procedures: None

TAD16F FLOW CHART

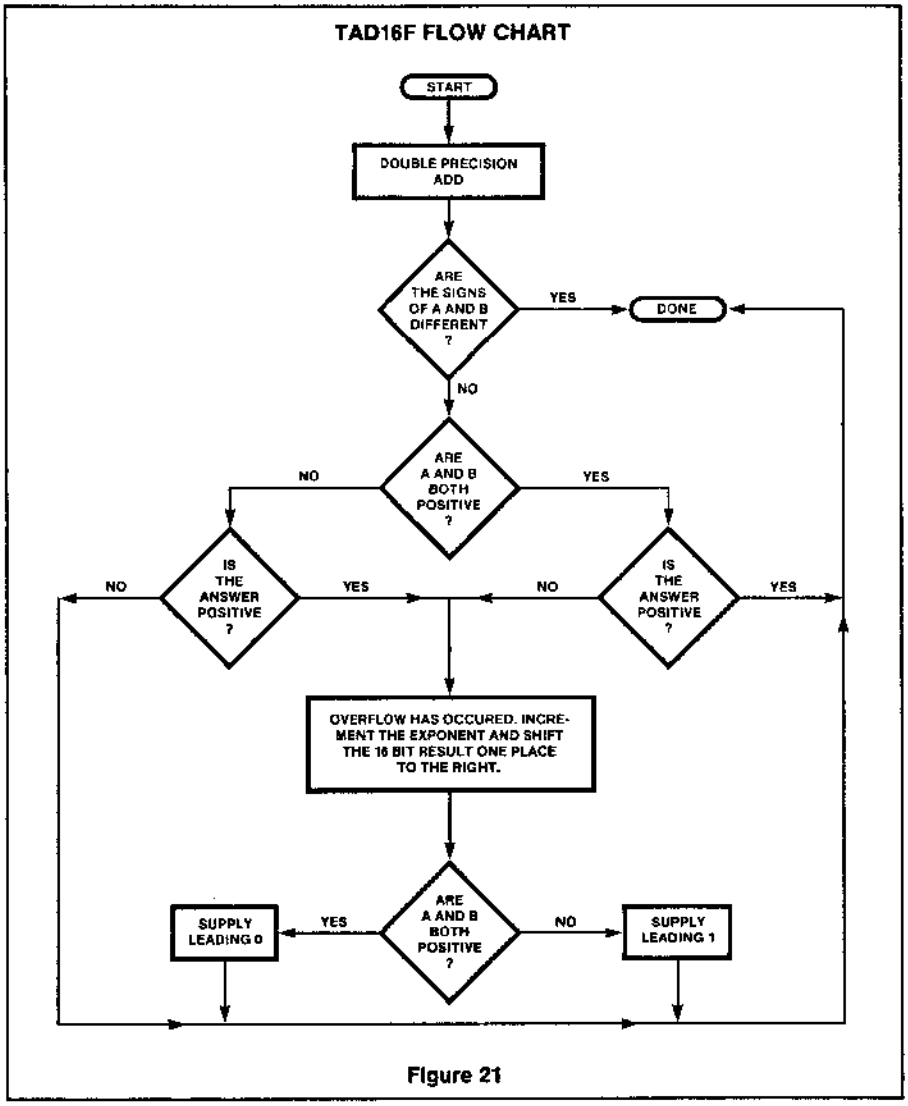


Figure 21

TAD16F PROGRAM LISTING

```
      PROC TAD16F

      MCLIB PROCEDURE TO ADD TWO DOUBLE PRECISION
      2's COMPLEMENT NUMBERS. NORMALLY USED WITH
      FPADD.

DMY RIV 200,7,8
SIGN RIV 200,0,1
L RIV 200,7,1
A RIV 200,6,8

NOTE NONE OF THE VARIABLES DECLARED IN THIS
PROCEDURE ARE EVER TRANSMITTED TO THE IVR
REG. THUS THEIR WS ADDRESSES ARE NOT IMPORTANT.

TAD       MOVE  R5,IVR      ENABLE HOB OF OP1.
          MOVE  SIGN,AUX
          MOVE  R6,IVR      ENABLE HOB OF OP2.
          ADD   SIGN,R4     R4 = 0 IF BOTH POS.
                            R4 = 1 IF BOTH DIFF.
                            R4 = 2 IF BOTH NEG.
          XMIT  1,AUX
          ADD   R6,R3       R3 POINTS TO LOB OF OP3.
          ADD   R5,IVR      ENABLE LOB OF OP1.
          MOVE  DMY,AUX
          MOVE  R3,IVR      ENABLE LOB OF OP2.
          ADD   DMY,DMY     OP1L + OP2L IN OP2L
          MOVE  OVF,AUX
          MOVE  R5,IVR      ENABLE HOB OF OP1.
          ADD   DMY,AUX
          MOVE  R6,IVR      ENABLE HOB OF OP2.
          ADD   DMY,DMY     ANSWER IN OP2.
          XEC   * + 1(R4)
          JMP   ZEROS
          JMP   INBOUNDS
          JMP   ONES
          ORG   16,32
ZEROS     NZT   SIGN,OVERFLOW
          JMP   INBOUNDS
ONES      NZT   SIGN,INBOUNDS
OVERFLOW  XMIT  1,AUX
          MOVE  R2,IVR      ENABLE X2 WHICH IS XA.
          ADD   DMY,DMY     INCREMENT EXPONENT.
          XMIT  2,AUX
          ADD   R2,R5       R5 POINTS TO LOB.
          MOVE  R6,IVR      R6 POINTS TO THE HOB.
          MOVE  L,R6        MOVE BIT 7 OF HOB TO R6.
          MOVE  A,DMY       ROTATE HOB.
          MOVE  OVF,SIGN    SHIFT IN A 0 OP 1.
          MOVE  R5,IVR      ENABLE LOB.
          MOVE  A,DMY       ROTATE LOB.
          MOVE  R6,SIGN     MOVE BIT 7 OF HOB TO
                            BIT 0 OF THE LOB.
INBOUNDS RTN
END TAD16F
```

Figure 22

DATA SHEETS

8T31-N,F

DESCRIPTION

The I/O Port is an 8-bit bidirectional data register designed to function as an I/O interface element in microprocessor systems. It contains 8 clocked data latches accessible from either a microprocessor port or a user port. Separate I/O control is provided for each port. The 2 ports operate independently, except when both are attempting to input data into the I/O Port. In this case, the user port has priority.

A master enable (ME) is provided that enables or disables the μP bus regardless of the state of the other inputs, but has no effect on the user bus.

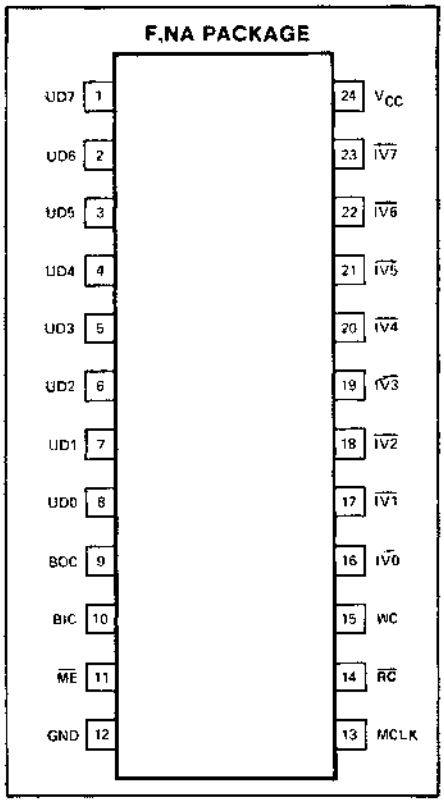
A unique feature of this family is its ability to start up in a predetermined state. If the clock is maintained at a voltage less than .8V until the power supply reaches 3.5V, the user port will always be all logic 1 levels, while the microprocessor port will be all logic 0 levels.

FEATURES

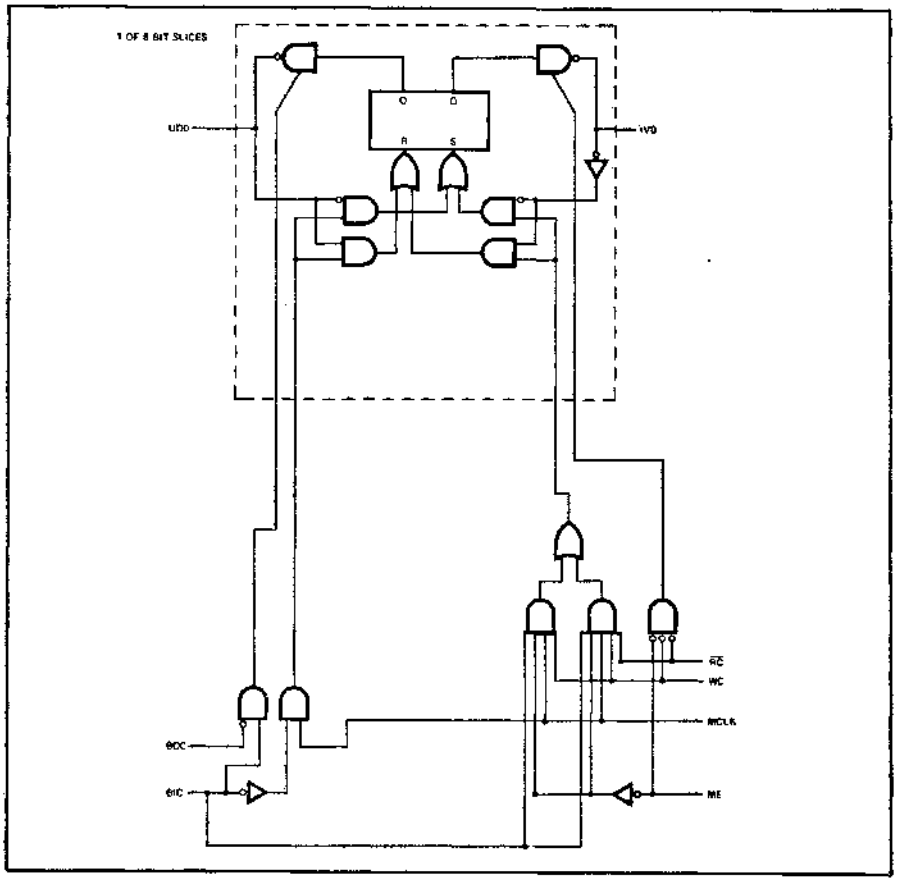
- **Each device has 2 ports, one to the user, the other to a microprocessor. I/O Ports are completely bidirectional**
- **Ports are independent, with the user port having priority for data entry**
- **User data input synchronous**
- **The user data bus is available with tri-state (8T32, 8T36) or open collector (8T33, 8T35) outputs**
- **At power up, the user port outputs are high**
- **Tri-state TTL outputs for high drive capability**
- **Directly compatible with the 8X300 Microcontroller**
- **Operates from a single 5V power supply over a temperature range of 0°C to +70°C**

PIN CONFIGURATION

F,NA PACKAGE



BLOCK DIAGRAM



PIN DESIGNATION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
1-8	UD0-UD7:	User Data I/O Lines. Bidirectional data lines to communicate with user's equipment.	Acitve high three-state
16-23	$\overline{IV0}$-$\overline{IV7}$:	Microprocessor Bus. Bidirectional data lines to communicate with controlling digital system.	Active low three-state
10	$\overline{BIC}$:	Input Control. User input to control writing into the I/O Port from the user data lines.	Active low
9	$\overline{BOC}$:	Output Control. User input to control reading from the I/O Port onto the user data lines.	Active low
11	$\overline{ME}$:	Master Enable. System input to enable or disable all other system inputs and outputs. It has no effect on user inputs and outputs.	Active low
15	WC:	Write Command. When WC is high, stores contents of IV0-IV7 as data.	Active high
14	$\overline{RC}$	Read Command. When RC is low, data is presented on IV0-IV7.	Active low
13	MCLK:	Master Clock. Input to strobe data into the latches. See function tables for details.	Active high
24	V_{CC}:	5V power connection.	
12	GND:	Ground.	

$\overline{BIC}$	$\overline{BOC}$	MCLK	USER DATA BUS FUNCTION
H	L	X	Output Data
L	X	H	Input Data
H	H	X	Inactive

H = High Level L = Low Level X = Don't care

Table 1 USER PORT CONTROL FUNCTION

$\overline{ME}$	$\overline{RC}$	WC	MCLK	$\overline{BIC}$	MICROPROCESSOR BUS FUNCTION
L	L	L	X	X	Output Data
L	X	H	H	H	Input Data
X	H	L	X	X	Inactive
X	X	H	X	L	Inactive
H	X	X	X	X	Inactive

Table 2 MICROPROCESSOR PORT CONTROL FUNCTION

USER DATA BUS CONTROL

The activity of the user data bus is controlled by the BIC and BOC inputs as shown in Table 1.

The user data input is a synchronous function with MCLK. A low level on the BIC input allows data on the user data bus to be written into the data latches only if MCLK is at a high level. A low level on the BIC input allows data on the user data bus to be latched regardless of the level of the MCLK input.

To avoid conflicts at the data latches, input from the microprocessor port is inhibited when BIC is at a low level. Under all other conditions the 2 ports operate independently.

MICROPROCESSOR BUS CONTROL

As is shown in Table 2, the activity of the microprocessor port is controlled by the ME, RC, WC and BIC inputs, as well as the state of an internal status latch. BIC is included to show user port priority over the microprocessor port for data input.

BUS OPERATION

Data written into the I/O Port from one port will appear inverted when read from the other port. Data written into the I/O Port from one port will not be inverted when read from the same port.

DC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ± 5%, 0°C ≤ T_A 70°C unless otherwise specified.

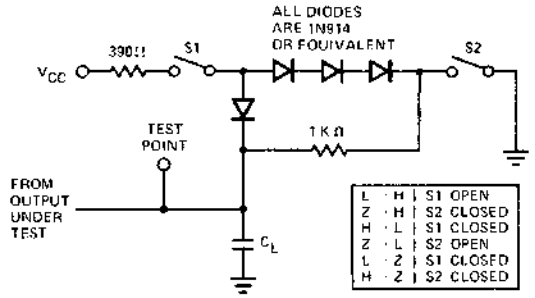
PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
	Input voltage					V
V_{IH}	High		2.0			
V_{IL}	Low				.8	
V_{IC}	Clamp	I_I = -5mA			-1	
	Output voltage	V_{CC} = 4.75V				V
V_{OH}	High		2.4			
V_{OL}	Low				.55	
	Input current[1]	V_{CC} = 5.25V				μA
I_{IH}	High	V_{IH} = 5.25V		<10	100	
I_{IL}	Low	V_{IL} = .5V		-350	-550	
	Output current[2]					mA
I_{OS}	Short circuit	V_{CC} = 4.75V				
	UD bus		10			
	IV bus		20			
I_{CC}	VCC supply current	V_{CC} = 5.25V		100	150	mA

NOTES

1. The input current includes the tri-state/open collector leakage current of the output driver on the data lines.
2. Only one output may be shorted at a time.

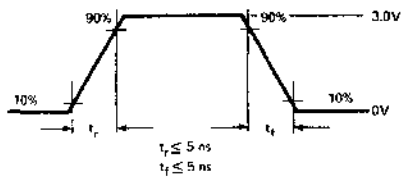
PARAMETER MEASUREMENT INFORMATION

LOAD CIRCUIT FOR TRI-STATE OUTPUTS

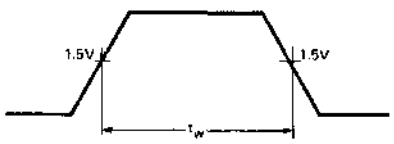


NOTE. C_L includes fixture capacitance

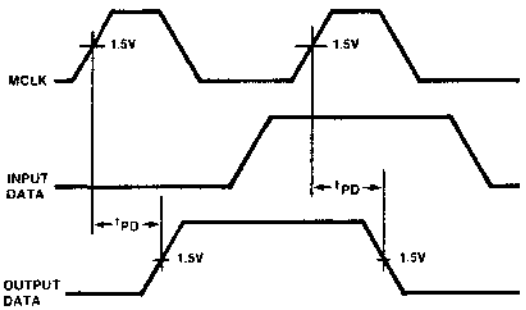
INPUT WAVEFORM



CLOCK PULSE WIDTH



DATA DELAY TIMES
Clock Referenced



AC ELECTRICAL CHARACTERISTICS $0°C \leq T_A \leq 70°C$, $V_{CC} = 5V \pm 5\%$

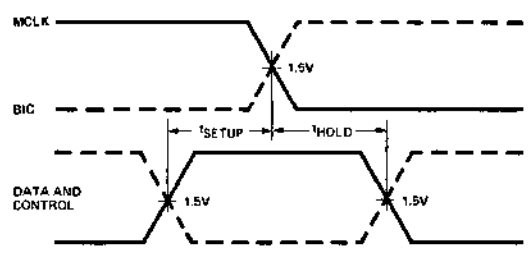
PARAMETER	INPUT	TEST CONDITION	LIMITS Min	LIMITS Typ	LIMITS Max	UNIT
t_{PD} User data delay[1]	UDX	$C_L = 50pF$		25	38	ns
	MCLK			45	61	ns
t_{OE} User output enable	BOC	$C_L = 50pF$	18	26	47	ns
t_{OD} User output disable	BIC	$C_L = 50pF$	18	28	35	ns
	BOC		16	23	33	ns
t_{PD} µP data delay[1]	IVBX	$C_L = 50pF$		38	53	ns
	MCLK			48	61	ns
t_{OE} µP output enable	ME RC WC	$C_L = 50pF$	14	19	25	ns
t_{OD} µP output disable	ME RC WC	$C_L = 50pF$	13	17	32	ns
t_W Minimum pulse width	MCLK		40			ns
t_{SETUP} Minimum setup time[2]	UDX[3]		15			ns
	BIC		25			
	IVX		55			
	ME		30			
	RC		30			
	WC		30			
t_{HOLD} Minimum hold time[2]	UDX[3]		25			ns
	BIC		10			
	IVX		10			
	ME		5			
	RC		5			
	WC		5			

NOTES

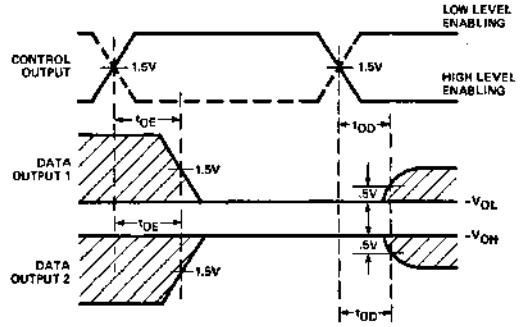
1. Data delays referenced to the clock are valid only if the input data is stable at the arrival of the clock and the hold time requirement is met.
2. Set up and hold times given are for "normal" operation. BIC setup and hold times are for a user write operation. RC setup and hold times are for an I/O Port select operation. ME and WC setup and hold times are for a microprocessor bus write operation.
3. Times are referenced to MCLK.

VOLTAGE WAVEFORMS

SETUP AND HOLD TIMES



OUTPUT ENABLE AND DISABLE TIMES
(Tri-State Outputs)



WAVEFORM #1 IS FOR AN OUTPUT WITH INTERNAL CONDITIONS SUCH THAT THE OUTPUT IS LOW WHEN THE TRI-STATE DRIVER IS ENABLED WAVEFORM #2 IS FOR THE OPPOSITE CONDITION.

8T32/8T33/8T35/8T36-F,NA

TYPES

8T32 Tri-State, Synchronous User Port
8T33 Open Collector, Synchronous User Port
8T35 Open Collector, Asynchronous User Port
8T36 Tri-State, Asynchronous User Port

DESCRIPTION

The Addressable I/O Port is an 8-bit bidirectional data register designed to function as an I/O interface element in microprocessor systems. It contains 8 data latches accessible from either a microprocessor port or a user port. Separate I/O control is provided for each port. The 2 ports operate independently, except when both are attempting to input data into the I/O Port. In this case, the user port has priority.

A unique feature of the I/O Port is the way in which it is addressed. Each device has an 8-bit, field programmable address, which is used to enable the microprocessor port. When the SC control signal is high, data at the microprocessor port is treated as an address. If the address matches the I/O Port's internally programmed address, the microprocessor port is enabled, allowing data transfer through it.

The port remains enabled until an address which does not match is presented, at which time the port is disabled (data transfer is inhibited). A Master Enable input (ME) can serve as a ninth address bit, allowing 512 I/O Ports to be individually selected on a bus, without decoding. The user port is accessible at all times, independent of whether or not the microprocessor port is selected.

A unique feature of this family is their ability to start up in a predetermined state. If the clock is maintained at a voltage less than .8V until the power supply reaches 3.5V, the user port will always be all logic 1 levels, while the port will be all logic 0 levels.

ORDERING

The 8T32/33/35/36 may be ordered in preaddressed form. To order a preaddressed device use the following part number format:

N8TYY-XXX P

- P = F Ceramic package, NA Plastic package
- XXX = Any address from 000 through 255 (decimal) - 256 available addresses
- YY = I/O Port version (32, 33, 35, 36)

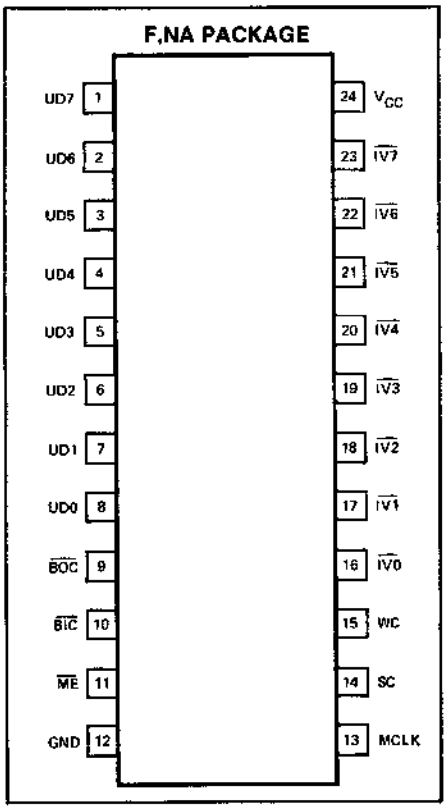
A stock of 8T32s and 8T36s with addresses 1 through 10 will be maintained. A small quantity of addresses 11 through 50 will also be available with a longer lead time.

FEATURES

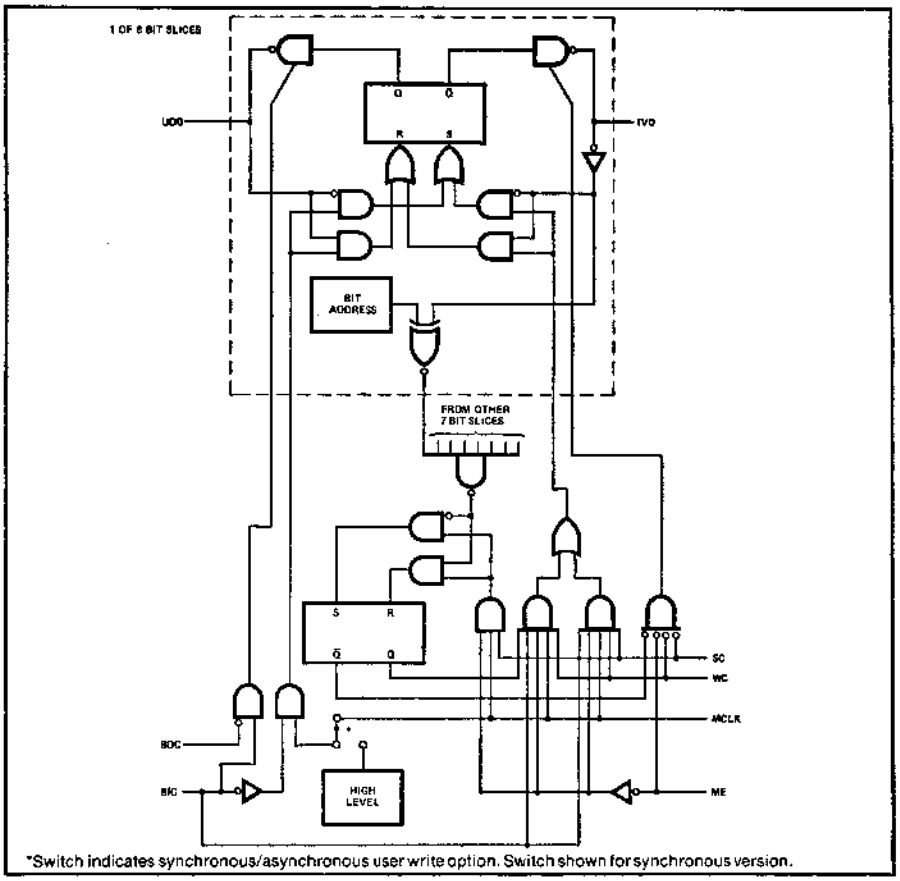
- **A field-programmable address allows 1 of 512 I/O Ports on a bus to be selected, without decoders.**
- **Each device has 2 ports, one to the user, the other to a microprocessor.**
- **Completely bidirectional.**
- **Ports are independent, with the user port having priority for data entry.**
- **A selected I/O Port de-selects itself when another I/O Port address is sensed.**
- **User data input available as synchronous (8T32, 8T33) or as asynchronous (8T35, 8T36) function.**
- **The user data bus is available with tri-state (8T32, 8T36) or open collector (8T33, 8T35) outputs.**
- **At power up, the I/O Port is not selected and the user port outputs are high.**
- **Tri-state TTL outputs for high drive capability.**
- **Directly compatible with the 8X300 Microcontroller.**
- **Operates from a single 5V power supply over a temperature range of 0° C to 70° C.**

PIN CONFIGURATION

F,NA PACKAGE



BLOCK DIAGRAM



*Switch indicates synchronous/asynchronous user write option. Switch shown for synchronous version.

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
1-8	UD0-UD7:	User Data I/O Lines. Bidirectional data lines to communicate with user's equipment. Either tri-state or open collector outputs are available.	Active high
16-23	$\overline{IV0}$-$\overline{IV7}$:	Microprocessor Bus. Bidirectional data lines to communicate with controlling digital system (microprocessor).	Active low three-state
10	$\overline{BIC}$:	Input Control. User input to control writing into the I/O Port from the user data lines.	Active low
9	$\overline{BOC}$:	Output Control. User input to control reading from the I/O Port onto the user data lines.	Active low
11	$\overline{ME}$:	Master Enable. System input to enable or disable all other system inputs and outputs. It has no effect on user inputs and outputs.	Active low
15	WC:	Write Command. When WC is high and SC is low, I/O Port, if selected, stores contents of IV0-IV7 as data.	Active high
14	SC:	Select Command. When SC is high and WC is low, data on IV0-IV7 is interpreted as an address. I/O Port selects itself if its address is identical to μP bus data; it de-selects itself otherwise.	Active high
13	MCLK:	Master Clock. Input to strobe data into the latches. See function tables for details.	Active high
24	VCC:	5V power connection.	
12	GND:	Ground.	

$\overline{BIC}$	$\overline{BOC}$	MCLK	USER DATA BUS FUNCTION	
			8T32, 8T33	8T35, 8T36
H	L	X	Output Data	Output Data
L	X	H	Input Data	Input Data
L	X	L	Inactive	Input Data
H	H	X	Inactive	Inactive

H = High Level L = Low Level X = Don't care

Table 1 USER PORT CONTROL FUNCTION

$\overline{ME}$	SC	WC	MCLK	$\overline{BIC}$	STATUS LATCH	I/O PORT FUNCTION
L	L	L	X	X	SET	Output Data
L	L	H	H	H	SET	Input Data
L	H	L	H	X	X	Input Address
L	H	H	H	L	X	Input Address
L	H	H	H	H	X	Input Data and Address
L	X	H	L	X	X	Inactive
L	H	X	L	X	X	Inactive
L	L	H	H	L	X	Inactive
L	L	X	X	X	Not Set	Inactive
H	X	X	X	X	X	Inactive

Table 2 MICROPROCESSOR PORT CONTROL FUNCTION

USER DATA BUS CONTROL

The activity of the user data bus is controlled by the BIC and BOC inputs as shown in Table 1.

For the 8T32 and 8T33, user data input is a synchronous function with MCLK. A low level on the BIC input allows data on the user data bus to be written into the data latches only if MCLK is at a high level. For the 8T35 and 8T36, user data input is an asynchronous function. A low level on the BIC input allows data on the user data bus to be latched regardless of the level of the MCLK input. Note that when 8T35 or 8T36 are used with the 8X300 Microcontroller care must be taken to insure that the Microprocessor bus is stable when it is being read by the 8X300 Microcontroller.

To avoid conflicts at the Data Latches, input from the Microprocessor Port is inhibited when BIC is at a low level. Under all other conditions the 2 ports operate independently.

MICROPROCESSOR BUS CONTROL

As is shown in Table 2, the activity of the microprocessor port is controlled by the ME, SC, WC and BIC inputs, as well as the state of an internal status latch. BIC is included to show user port priority over the microprocessor port for data input.

Each I/O Port's status latch stores the result of the most recent I/O Port select; it is set when the I/O Port's internal address matches the Microprocessor Bus. It is cleared when an address that differs from the internal address is presented on the Microprocessor Bus.

In normal operation, the state of the status latch acts like a master enable; the microprocessor port can transfer data only when the status latch is set.

When SC and WC are both high, data on the Microprocessor Bus is accepted as data, whether or not the I/O Port was selected. The data is also interpreted as an address. The I/O Port sets its select status if its address matches the data read when SC and WC were both high; it resets its select status otherwise.

BUS OPERATION

Data written into the I/O Port from one port will appear inverted when read from the other port. Data written into the I/O Port from one port will not be inverted when read from the same port.

AC ELECTRICAL CHARACTERISTICS $0°C \leq T_A \leq 70°C$, $V_{CC} = 5V \pm 5\%$

PARAMETER	INPUT	TEST CONDITION	LIMITS			UNIT
			Min	Typ	Max	
t_{PD} User data delay (Note 1)	UDX MCLK* BIC†	C_L = 50pF		25 45 40	38 61 55	ns
t_{OE} User output enable	BOC	C_L = 50pF	18	26	47	ns
t_{OD} User output disable	BIC BOC	C_L = 50pF	18 16	28 23	35 33	ns
t_{PD} µP data delay (Note 1)	IVBX MCLK	C_L = 50pF		38 48	53 61	ns
t_{OE} µP output enable	ME SC WC	C_L = 50pF	14	19	25	ns
t_{OD} µP output disable	ME SC WC	C_L = 50pF	13	17	32	ns
t_W Minimum pulse width	MCLK BIC†		40 35			ns
t_{SETUP} Minimum setup time	UDX□ BIC* IVX ME SC WC	(Note 2)	15 25 55 30 30 30			ns
t_{HOLD} Minimum hold time	UDX□ BIC* IVX ME SC WC	(Note 2)	25 10 10 5 5 5			ns

* Applies for 8T32 and 8T33 only.
† Applies for 8T35 and 8T36 only
□ Times are referenced to MCLK for 8T32 and 8T33, and are referenced to BIC for 8T35 and 8T36.

NOTES:

1. Data delays referenced to the clock are valid only if the input data is stable at the arrival of the clock and the hold time requirement is met.
2. Set up and hold times given are for "normal" operation. BIC setup and hold times are for a user write operaton. SC setup and hold times are for an I/O Port select operation. WC setup and hold times are for an Microprocessor Bus write operation. ME setup and hold times are for both IV write and select operations

DC ELECTRICAL CHARACTERISTICS $0°C \le T_A \le 70°C$, $V_{CC} = 5V \pm 5\%$

PARAMETER		TEST CONDITIONS	LIMITS			UNITS
			Min	Typ	Max	
V_{IH}	High-level input voltage		2.0		5.5	V
V_{IL}	Low-level input voltage		-1.0		.8	V
V_{CL}	Input clamp voltage	I_I = -5mA			-1	V
I_{IH}	High-level input current[1]	V_{CC} = 5.25V V_{IH} = 5.25V		<10	100	μA
I_{IL}	Low level input current[1]	V_{CC} = 5.25V V_{IL} = .5V		-350	-550	μA
V_{OL}	Low-level output voltage	V_{CC} = 4.75V I_{OL} = 16mA			.55	V
V_{OH}	High-level output voltage	V_{CC} = 4.75V I_{OH} = -3.2mA	2.4			V
I_{OS}	Short-circuit output current[2] UD bus IV bus	 V_{CC} = 4.75V V_{CC} = 4.75V	 10 20			 mA mA
I_{CC}	Supply current	V_{CC} = 5.25V		100	150	mA

NOTES

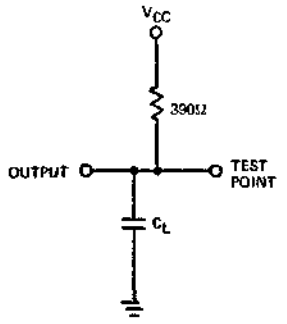
1. The input current includes the Tri-state/Open Collector leakage current of the output driver on the data lines.
2. Only one output may be shorted at a time.
3. These limits do not apply during address programming.

Absolute Maximum Ratings:

Supply voltage[3] 7V

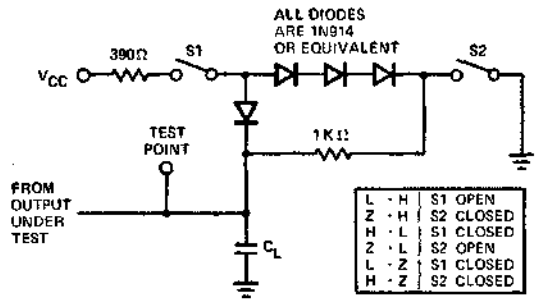
Input voltage[3] 5.5V

TEST LOAD CIRCUIT (OPEN COLLECTOR OUTPUTS)



NOTE: C_L includes fixture capacitance.

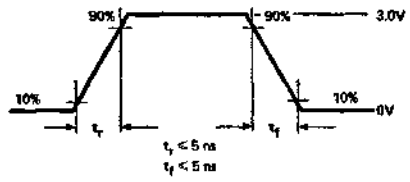
TEST LOAD CIRCUIT (TRI-STATE OUTPUTS)



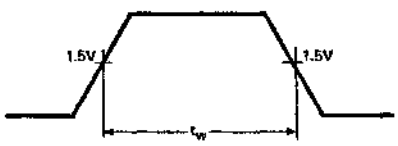
NOTE: C_L includes fixture capacitance.

VOLTAGE WAVEFORMS

INPUT WAVEFORM

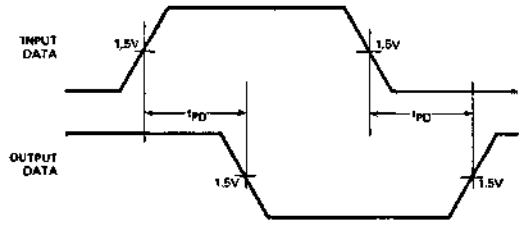


CLOCK PULSE WIDTH

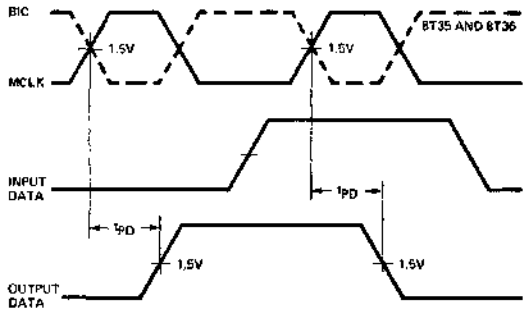


VOLTAGE WAVEFORMS (Cont'd)

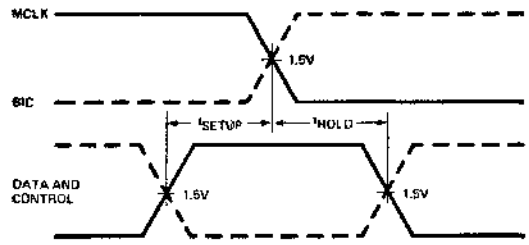
DATA DELAY TIMES
Input Data Reference



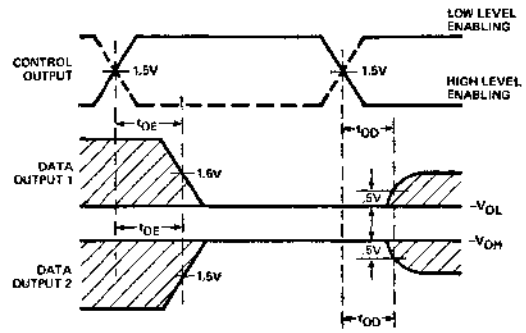
DATA DELAY TIMES
Clock Referenced



SETUP AND HOLD TIMES



OUTPUT ENABLE AND DISABLE TIMES
(Tri-State Outputs)



Waveform #1 is for an output with internal conditions such that the output is Low when the tri-state driver is enabled. Waveform #2 is for the opposite condition.

ADDRESS PROGRAMMING

The I/O Port is manufactured such that an address of all high levels (>2V) on the Microprocessor Bus inputs matches the Port's internal address. To program a bit so a low-level input (<0.8V) matches, the following procedure should be used:

1. Set all control inputs to their inactive state (BIC = BOC = ME = V_{CC}, SC = WC = MCLK = GND). Leave all Microprocessor Bus I/O pins open.
2. Raise V_{CC} to 7.75V ± .25V.
3. After V_{CC} has stabilized, apply a single programming pulse to the user data bus bit where a low-level match is desired. The voltage should be limited to 18V; the current should be limited to 75mA. Apply the pulse as shown in Figure 1.
4. Return V_{CC} to 0V. (Note 1).
5. Repeat this procedure for each bit where a low-level match is desired.
6. Verify that the proper address is programmed by setting the Port's status latch (IV0-IV7 = desired address, ME = WC = L, SC = MCLK = H). If the proper address has been programmed, data presented at the μP bus will appear inverted on the user bus outputs. (Use normal V_{CC} and input voltage for verification.)

After the desired address has been programmed, a second procedure must be followed to isolate the address circuitry. The procedure is:

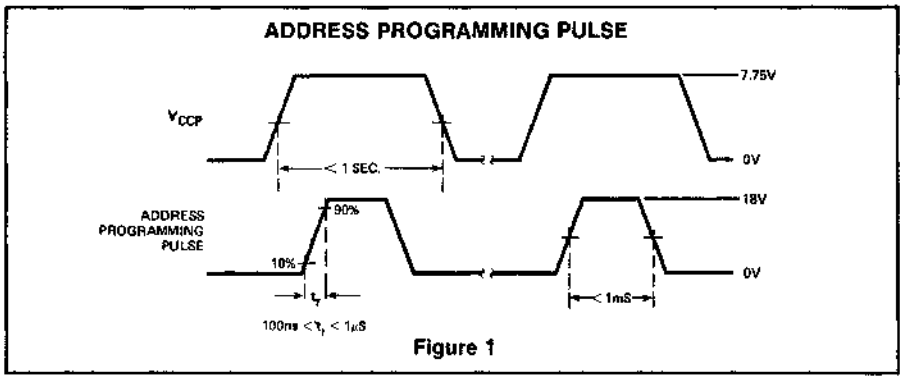


Figure 1

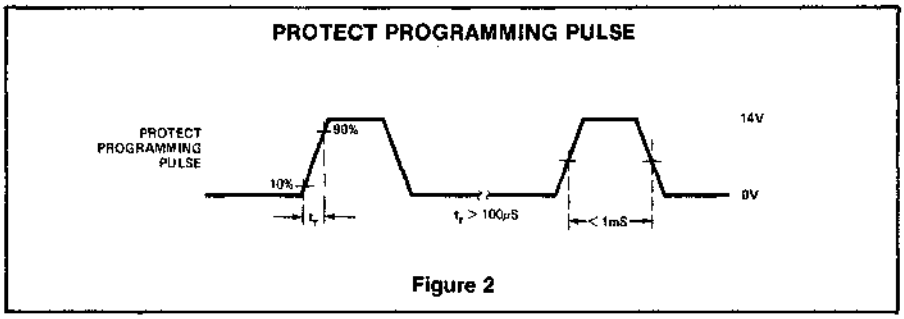


Figure 2

1. Set V_{CC} and all control inputs to 0V. (V_{CC} = BIC = BOC = ME = SC = WC = MCLK = 0V). Leave all Microprocessor Bus I/O pins open.
2. Apply a protect programming pulse to every user data bus pin, one at a time. The voltage should be limited to 14V; the current should be limited to 150mA. Apply the pulse as shown in Figure 2.
3. Verify that the address circuitry is isolated by applying 7V to each user data bus pin and measuring less than 1mA of input current. The conditions should be the same as in step 1 above. The rise time on the verification voltage must be slower than 100μs.

PROGRAMMING SPECIFICATIONS[1]

PARAMETER		TEST CONDITIONS	LIMITS			UNITS
			Min	Typ	Max	
V_{CCP}	Programming supply voltage					
	Address		7.5		8.0	V
	Protect			0		V
I_{CCP}	Programming supply current	V_{CCP} = 8.0V			250	mA
	Max time V_{CCP} > 5.25V				1.0	s
	Programming voltage					
	Address		17.5		18.0	V
	Protect		13.5		14.0	V
	Programming current					
	Address				75	mA
	Protect				150	mA
	Programming pulse rise time					
	Address		.1		1	μs
	Protect		100			μs
	Programming pulse width		.5		1	ms

NOTE

1. If all programming can be done in less than 1 second, V_{CC} may remain at 7.75V for the entire programming cycle.

APPLICATIONS

Figure 3 shows some of the various ways to use the I/O Port in a system. By controlling the BIC and BOC lines, the device may be used for the input and output of data, control, and status signals. I/O Port 1 functions bidirectionally for data transfer and I/O Port 2 provides a similar function for discrete status and control lines. I/O Ports 3 and 4 serve as dedicated output and input ports, respectively.

8T32 APPLICATIONS

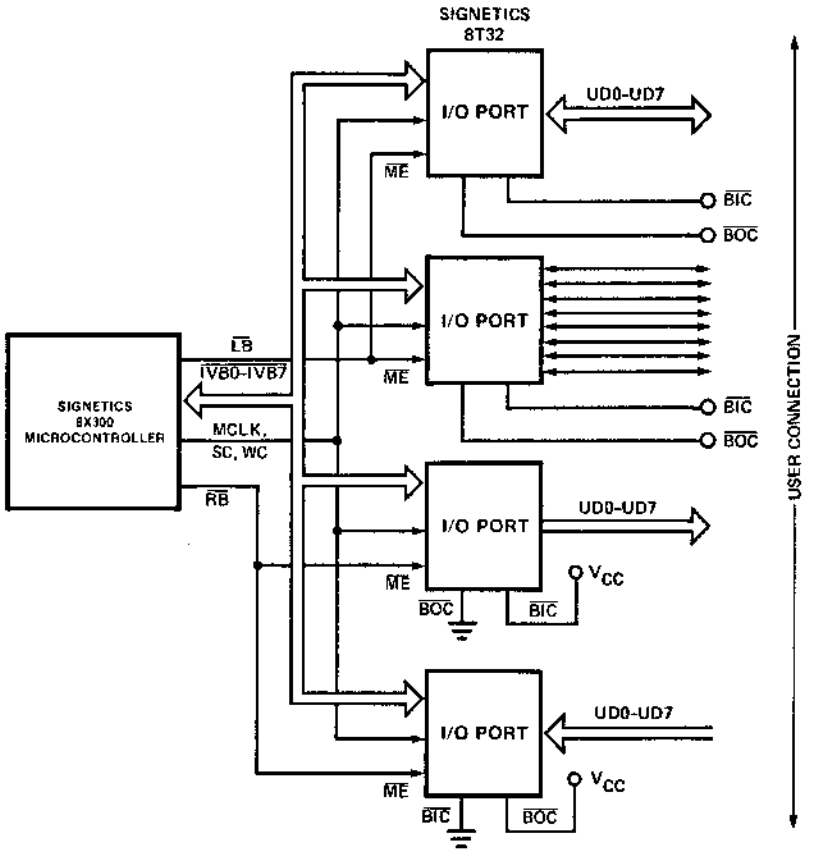


Figure 3

DESCRIPTION

The Bus Expander is specifically designed to increase the I/O capability of 8X300 systems previously limited by fanout considerations. The bus expander serves as a buffer between the 8X300 and blocks of I/O devices. Each bus expander can buffer a block of 16 I/O ports while only adding a single load to the 8X300.

FEATURES

- **15ns max propagation delay**
- **Bidirectional**
- **Three-state outputs on both ports**
- **Pre-programmed address range**

APPLICATIONS

The 8T39 Bus Expander is designed to be used with the 8X300 microprocessor to allow increased I/O capability in those systems previously limited by fanout considerations. Figure 1 shows a typical arrangement of the bus expander in an 8X300 system. Each expander services I/O ports whose address is within the range of the expander. Other I/O ports or working storage may be directly connected to the bus as shown.

The bus expander is not limited to use with the 8X300, but may be applied in any system which uses a combined address/data bus.

8T39 ADDRESSING

During normal operation of the 8X300 when an I/O port address is being sent on the IV Bus (SC is high), the I/O port will examine all eight bits of the microprocessor bus for an address compare. Since the 8T39 is used to buffer blocks of I/O ports, only the four most significant bits are examined by the 8T39 for an address compare.

Note that redundant addresses are not programmed into separate devices. Rather, a discrete device (such as the 8T39-03) may be wired for any address requiring two 1 bits and two 0 bits in the address. The various address ranges for this same device are obtained by permuting the high order (DI0 and DO0 are MSB) data lines accordingly. Both input and output lines must be redefined in order to maintain data and address integrity on the extended bus. Table 1 summarizes the 8T39 addressing.

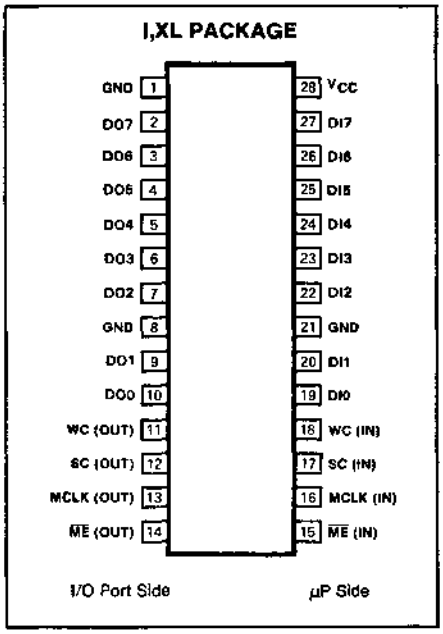
Address functions are specified with the convention that bit 0 is the MSB and bit 7 is the LSB. The DI microprocessor bus address decoding is active low.

FUNCTIONAL DESCRIPTION

The Bus Expander contains eight sets of non-inverting bidirectional tri-state drivers for the bus data bits, four non-inverting unidirectional drivers for I/O port control, and necessary control logic. The control logic is required to maintain the proper directional transfer of bus data as dictated by the states of the I/O port control signals and the currently enabled I/O port. Each bus expander is programmed during manufacturing to respond to a specific block of I/O port addresses. Only I/O ports with addresses in the range of a given bus expander may be connected to that expander. A bus expander may be used on either left bank or right bank. Multiple expanders on the same bank must have different address ranges; however, expanders with the same address range can be connected if they are on different banks. Systems may be configured with I/O ports connected directly to the 8X300, as well as I/O ports connected through a bus expander; however, no unbuffered I/O port may have an address within the span of a bus expander on the same bank.

Addition of bus expanders may impact system cycle time due to the added delay in the data path. For the purposes of calculating allowable cycle time as described in the 8X300 data sheet, the bus expander delays may be considered additive to the I/O port delays so that a buffered I/O port simply appears as a slower I/O port.

PIN CONFIGURATION



PIN DESIGNATION

PIN NO.	SYMBOL	NAME AND FUNCTION	TYPE
2-7,9,10	DO0-DO7	I/O port data bus	Active low, three-state
11	WC(OUT)	Write command output	Active high
12	SC(OUT)	Select command output	Active high
13	MCLK(OUT)	Master clock output	Acitve high
14	$\overline{ME}$(OUT)	Master enable output	Active low
15	$\overline{ME}$(IN)	Master enable input	Active low
16	MCLK(IN)	Master clock input	Active high
17	SC(IN)	Select command output	Active high
18	WC(IN)	Write command output	Active high
19,20,22-27	DI0-DI7	Microprocessor data bus	Active low, three-state
1,8,21	GND	Ground	
28	V_{CC}	+5 volt supply	

PART TYPE	ADDRESS PATTERN MSB(0) LSB(7)	ADDRESS BLOCKS Octal
8T39-00	0000XXXX	0-17
8T39-01	0001XXXX	20-37, 40-57, 100-117, 200-217
8T39-03	0011XXXX	60-77, 120-137, 220-237, 140-157, 240-257, 300-317
8T39-07	0111XXXX	160-177, 260-277, 320-337, 340-357
8T39-17	1111XXXX	360-377

Table 1 8T39 ADDRESSING SUMMARY

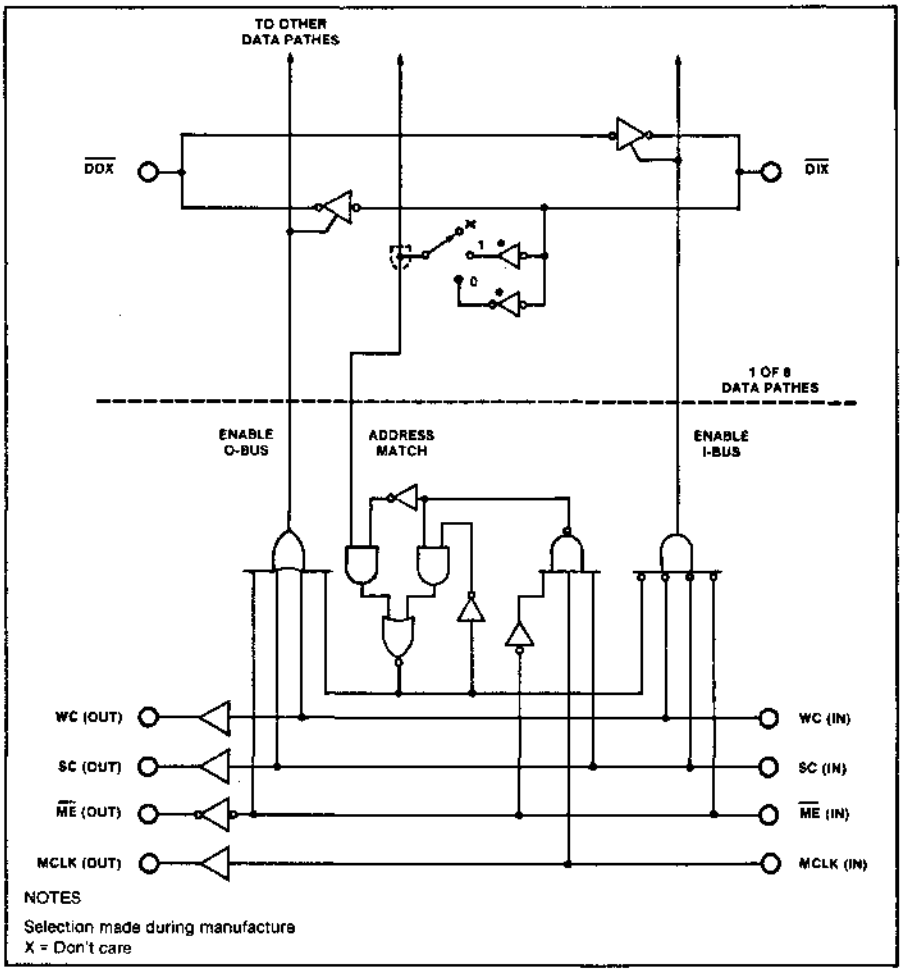
TRUTH TABLE

$\overline{ME}$	SC	WC	MCLK	SELECT LATCH	DATA TRANSFER DIRECTION	ADDRESS* COMPARISON
L	L	L	X	Set	DI Bus ← DO Bus	No
L	L	L	X	Not set	DI Bus → DO Bus	No
L	L	H	X	X	DI Bus → DO Bus	No
L	H	X	L	X	DI Bus → DO Bus	No
L	H	X	H	X	DI Bus → DO Bus	Yes
H	X	X	X	X	DI Bus - DO Bus	No

NOTES

*When an address comparison is made, the select latch is set if the data on the DI Bus is within the manufactured address range of the IV Bus Expander. Otherwise, the select latch is cleared.

FUNCTIONAL BLOCK DIAGRAM



NOTES

Selection made during manufacture
X = Don't care

ABSOLUTE MAXIMUM RATINGS[1]

PARAMETER		RATING	UNIT
V_{CC}	Power supply voltage	+7	Vdc
V_{IN}	Input voltage	+5.5	Vdc
V_O	Off-state output voltage	+5.5	Vdc
T_A	Operating temperature range	0 to +70	°C
T_{STG}	Storage temperature range	-65 to +150	°C

ORDERING INFORMATION

The Bus Expander is ordered by specifying the following part number:

N8T39-XX P

P = { I - Ceramic Package; XL - Epoxy Package

XX = Address Range As Determined From Table 1

DC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ± 5%, 0°C ≤ T_A ≤ 70°C

PARAMETER		TEST CONDITIONS	LIMITS Min	LIMITS Typ	LIMITS Max	UNIT
	Input voltage					V
V_{IL}	Low				.8	
V_{IH}	High	-5mA at V_{CC} min	2.0			
V_{IC}	Clamp				-1	
	Output voltage	V_{CC} = 4.75V				V
V_{OL}	Low	I_{OL} = 16mA			.55	
V_{OH}	High	I_{OH} = -3.2mA	2.4			
	Input current	V_{CC} = 5.25V				uA
I_{IL}	Low[1]	V_{IL} = .5V			-250	
I_{IH}	High[1]	V_{IH} = 5.25V		< 10	100	
I_{OS}	Short circuit output current	V_{CC} = 4.75V	-40			mA
I_{CC}	Supply current	V_{CC} = 5.25V			200	mA

AC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ± 5%, 0°C ≤ T_A ≤ 70°C, C_L = 300pF[2]

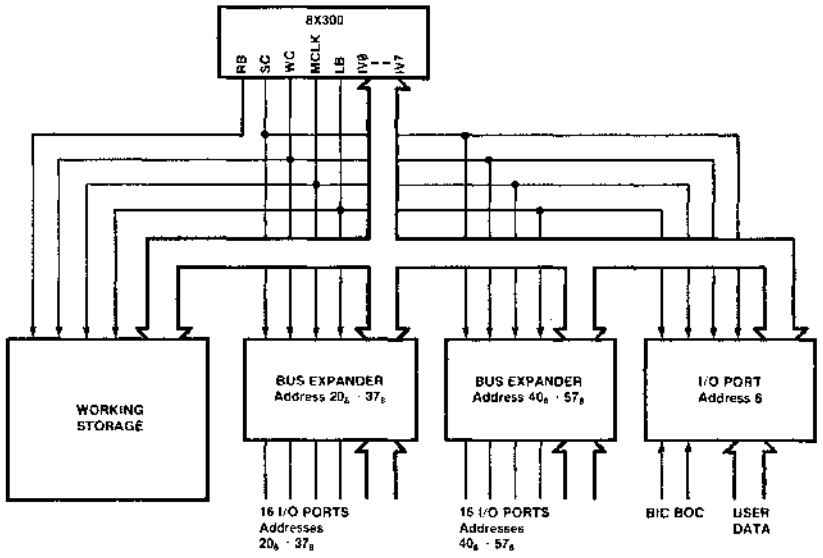
PARAMETER		TO	FROM	TEST CONDITIONS	LIMITS Min	LIMITS Typ	LIMITS Max	UNIT
t_{pd}	Propagation Delay Data	DOX DIX	DIX DOX				15	ns
t_{pd}	Control Propagation Delay	$\overline{ME}$ (out) MCLK (out) SC (out) WC (out)	$\overline{ME}$ (in) MCLK (in) SC (in) WC (in)				15	
t_{oe}	Data Output Enable	DIX DOX	$\overline{ME}$ (in) SC (in) WC (in)		28		56	ns
t_{od}	Data Output Disable	DIX DOX	$\overline{ME}$ (in) SC (in) WC (in)		15			ns
t_{setup}	Adverse Setup Time[3]	DIX DOX	DIX $\overline{ME}$ (in) MCLK (in) SC (in) WC (in)		54			ns
t_{hold}	Address Hold Time[3]	DIX DOX	DIX $\overline{ME}$ (in) MCLK (in) SC (in) WC (in)		3			ns

NOTES

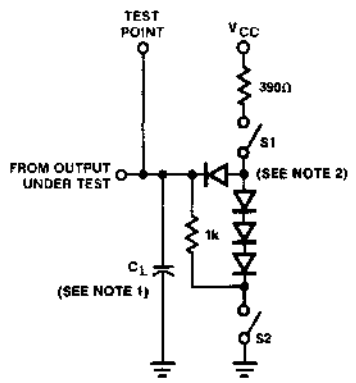
1. Includes tri-state leakage.
2. Minimum clock width ≈ 50ns.
3. All set up and hold times are referenced to the trailing edge of the clocking input MCLK.

TYPICAL APPLICATIONS

USING 2 BUS EXPANDERS TO GIVE 33 I/O PORTS PLUS WORKING STORAGE



TEST LOAD CIRCUIT



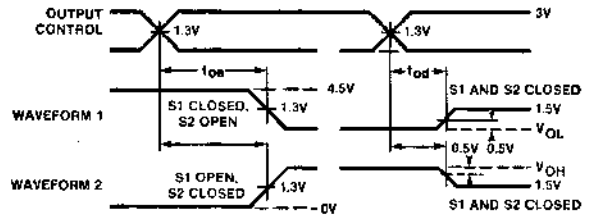
All resistors values are typical and in ohms.

NOTES

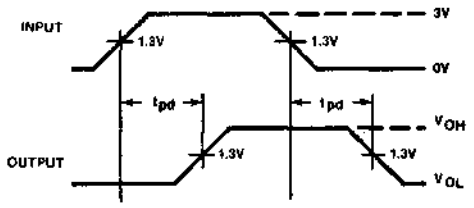
1. C_L includes probe and jig capacitance.
2. All diodes are 1N916 or 1N3064.

VOLTAGE WAVEFORMS

PROPAGATION DELAY TO THREE-STATE OUTPUTS



PROPAGATION DELAY TIMES



PRELIMINARY SPECIFICATION
Manufacturer reserves the right to make design changes and improvements.

8T58-I,XL

DESCRIPTION

The Bus Expander is specifically designed to increase the I/O capability of 8X300 systems previously limited by fanout considerations. The bus expander serves as a buffer between the 8X300 and blocks of I/O devices. Each bus expander can buffer a block of 16 I/O ports while only adding a single load to the 8X300.

FEATURES

- **15ns max propagation delay**
- **Bidirectional**
- **Three-state outputs on both ports**

FUNCTIONAL DESCRIPTION

The Bus Expander contains eight sets of non-inverting bidirectional tri-state drivers for the bus data bits, four non-inverting undirectional drivers for I/O port control, and necessary control logic. The control logic is required to maintain the proper directional transfer of bus data as dictated by the states of the I/O port control signals. A bus expander may be used on either left bank or right bank. Systems may be configured with I/O ports connected directly to the 8X300, as well as I/O ports connected through a bus expander.

Addition of bus expanders may impact system cycle time due to the added delay in the data path. For the purposes of calculating allowable cycle time as described in the 8X300 data sheet, the bus expander delays may be considered additive to the I/O port delays so that a buffered I/O port simply appears as a slower I/O port.

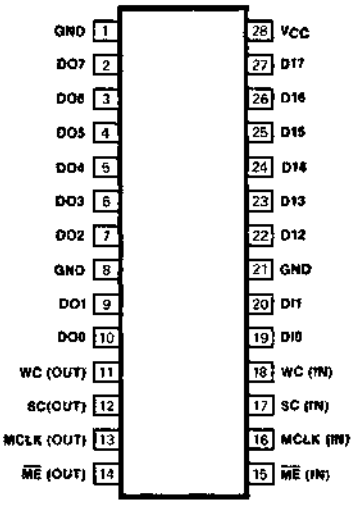
APPLICATIONS

The 8T39 Bus Expander is designed to be used with the 8X300 microprocessor to allow increased I/O capability in those systems previously limited by fanout considerations. Figure 1 shows a typical arrangement of the bus expander in an 8X300 system. Other I/O ports or working storage may be directly connected to the bus as shown.

The bus expander is not limited to use with the 8X300, but may be applied in any system which uses a combined address/data bus.

PIN CONFIGURATION

I,XL PACKAGE



μP Side

TRUTH TABLE

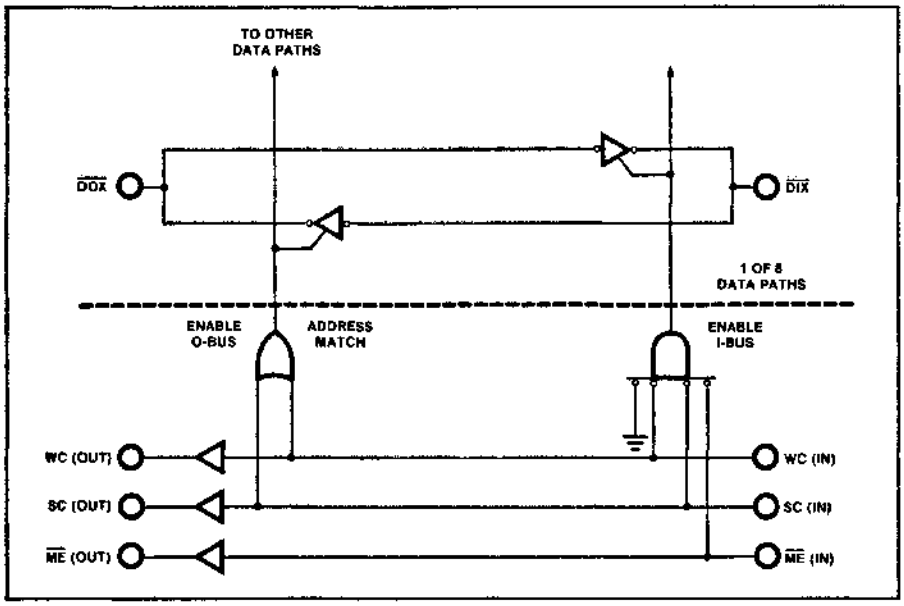
$\overline{ME}$	SC	WC	DATA TRANSFER DIRECTION	ADDRESS COMPARISON
L	L	L	DI Bus ← DO Bus	No
L	L	H	DI Bus → DO Bus	No
L	H	X	DI Bus → DO Bus	No
H	X	X	DI Bus → DO Bus	No

PIN DESIGNATION

PIN NO.	SYMBOL	NAME & FUNCTION	TYPE
2-7,9,10	DO0-DO7	I/O port data bus	Active low, three-state
11	WC(OUT)	Write command output	Active high
12	SC(OUT)	Select command output	Active high
13	MCLK(OUT)	Master clock input	Active high
14	$\overline{ME}$(OUT)	Master enable output	Active low
15	$\overline{ME}$(IN)	Master enable input	Active low
16	MCLK(IN)	Master clock input	Active high
17	SC(IN)	Select command output	Active high
18	WC(IN)	Write command output	Active high
19,20,22-27	DI0-DI7	Microprocessor data bus	Active low, three-state
1,8,21	GND	Ground	
28	V_{CC}	+5 volt supply	

PRELIMINARY SPECIFICATION

FUNCTIONAL BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

PARAMETER		RATING	UNIT
V_{CC}	Power supply voltage	+7	Vdc
V_{IN}	Input voltage	+5.5	Vdc
V_O	Off-state output voltage	+5.5	Vdc
T_A	Operating temperature range	0 to +70	°C
T_{STG}	Storage temperature range	-65 to +150	°C

AC ELECTRICAL CHARACTERISTICS $V_{CC} = 5V \pm 5\%$, $0°C \leq T_A \leq 70°C$, $C_L = 300pF$

PARAMETER		TO	FROM	TEST CONDITIONS	LIMITS			UNIT
					Min	Typ	Max	
t_{pd}	Path delay Data	DOX DIX	DIX DOX				15	ns
t_{pd}	Control	$\overline{ME}$(OUT) MCLK(OUT) SC(OUT) WC(OUT)	$\overline{ME}$(IN) MCLK(IN) SC(IN) WC(IN)				15	ns
t_{oe}	Data Output Enable	DIX DOX	$\overline{ME}$(IN) SC(IN) WC(IN)		28		56	ns
t_{od}	Data Output Disable	DIX DOX	$\overline{ME}$(IN) SC(IN) WC(IN)		15			

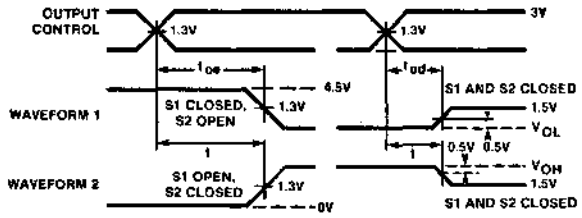
DC ELECTRICAL CHARACTERISTICS V_{CC} = 5V ±5%, 0°C ≤ T_A ≤ 70°C

PARAMETER		TEST CONDITIONS	LIMITS Min	LIMITS Typ	LIMITS Max	UNIT
	Input voltage					V
V_{IL}	Low				.8	
V_{IH}	High		2.0			
V_{IC}	Clamp	-5mA at V_{CC} min			-1	
	Output voltage	V_{CC} = 4.75V				V
V_{OL}	Low	I_{OL} = 50mA			.55	
V_{OH}	High	I_{OH} = -3.2mA	2.4			
	Input current	V_{CC} = 5.25V				μA
I_{IL}	Low*	V_{IL} = .5V			-250	
I_{IH}	High*	V_{IH} = 5.25V		<10	100	
I_{OS}	Short circuit output current	V_{CC} = 4.75V	-40			mA
I_{CC}	Supply current	V_{CC} = 5.25V			200	mA

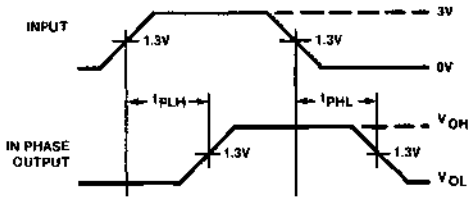
*Includes 3-State leakage.

VOLTAGE WAVEFORMS

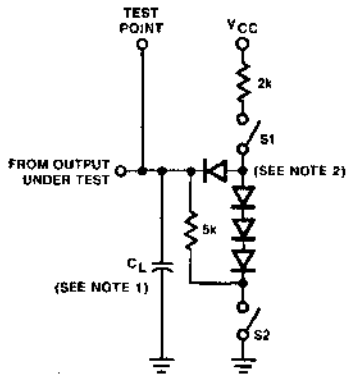
PROPAGATION DELAY TO THREE-STATE OUTPUTS



PROPAGATION DELAY TIMES



TEST LOAD CIRCUIT



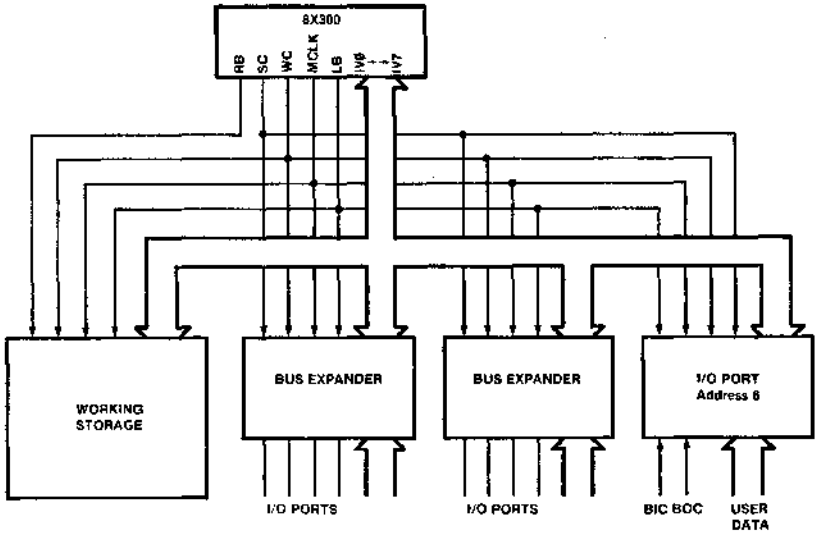
All resistors values are typical and in ohms.

NOTES

1. C_L includes probe and jig capacitance.
2. All diodes are 1N916 or 1N3064.

TYPICAL APPLICATION

USING 2 BUS EXPANDERS TO CREATE 33 I/O PORTS PLUS WORKING STORAGE



BIPOLAR MEMORY SELECTION GUIDE

DEVICE	ORGANI-ZATION	OUTPUT CIRCUIT[1]	OUTPUT LOGIC[2]	ACCESS TIME [ns][4]	TEMPERATURE RANGE[3]	PACKAGE	NO. OF PINS	MAX. I_{CC} [mA][4]
CAMS								
10155	8X2	OE	—	13	C	F,N	18	140
SAMS								
82S12	8X4	OC	T	40	C	F,N	24	160
82S112	8X4	TS	T	40	C	F,N	24	160
RAMS								
82S25	16X4	OC	B	50	M,C	F,N	16	105
3101A	16X4	OC	B	35	M,C	F,N	16	105
54/74S89	16X4	OC	T	50	M,C	F,N	16	105
54/74S189	16X4	TS	B	35	M,C	F,N	16	110
82S21	32X2	OC	T	50	C	F,N	16	130
82S16	256X1	TS	T	50	M,C	F,N	16	115
82S116	256X1	TS	T	40	C	F,N	16	115
82S17	256X1	OC	T	50	M,C	F,N	16	115
82S117	256X1	OC	T	40	C	F,N	16	115
54/74S200	256X1	TS	B	50	M,C	F,N	16	130
54/74S201	256X1	TS	B	50	M,C	F,N	16	130
54/74S301	256X1	OC	B	50	M,C	F,N	16	130
82S09	64X9	OC	T	45	M,C	I,N	28	190
82S10	1024X1	OC	B	45	M,C	F,N	16	170
82S110	1024X1	OC	B	35	C	F,N	16	170
82S11	1024X1	TS	B	45	M,C	F,N	16	170
82S111	1024X1	TS	B	35	C	F,N	16	170
93415A	1024X1	OC	B	45	M,C	F,N	16	170
93425A	1024X1	TS	B	45	M,C	F,N	16	170
82S208*	256X8	TS	B	60	C	F	22	185
82S210*	256X9	TS	B	60	C	F,N	24	185
82S400*	4096X1	OC	B	70	C	I	18	155
82S401*	4096X1	TS	B	70	C	I	18	155
ROMS								
82S226	256X4	OC	—	50	M,C	F,N	16	120
82S229	256X4	TS	—	50	M,C	F,N	16	120
82S214	256X8	TS	—	60	M,C	F,N	24	175
82S230	512X4	OC	—	50	M,C	F,N	16	140
82S231	512X4	TS	—	50	M,C	F,N	16	140
82S215	512X8	TS	—	60	M,C	F,N	24	175
82S240	512X8	OC	—	60	M,C	F,N	24	175
82S241	512X8	TS	—	60	M,C	F,N	24	175
8228	1024X4	TTL	—	50	C	F	16	170
82S280	1024X8	OC	—	70	M,C	F,N	24	140
82S281	1024X8	TS	—	70	M,C	F,N	24	140
82S290	2048X8	OC	—	80	M,C	F,N	24	170
82S291	2048X8	TS	—	80	M,C	F,N	24	170

*To be announced

NOTES

1. Output circuit:
   OE = Open emitter
   OC = Open collector
   TS = Tri-state
2. Output logic:
   T - Transparent—input data appears on output during Write
   B = Blanked—output is blanked during Write
3. Temperature range:
   C = Commercial (0° C to +75° C)
   M = Military (–55° C to +125° C)
   All ECL 10,000 series (–30° C to +85° C)
4. Commercial (0°C to +75°C)

BIPOLAR MEMORY SELECTION GUIDE (Cont'd)

DEVICE	ORGANI-ZATION	OUTPUT CIRCUIT[1]	OUTPUT LOGIC[2]	ACCESS TIME (ns)	TEMPERATURE RANGE[3]	PACKAGE	NO. OF PINS	MAX. I_{CC} (mA)	EQUIVA-LENT ROM
PROMS									
82S23	32X8	OC	—	50	M,C	F,N	16	77	—
82S123	32X8	TS	—	50	M,C	F,N	16	77	—
10139	32X8	OE	—	15	C	F,N	16	145	—
82S27	256X4	OC	—	40	C	F	16	140	—
82S126	256X4	OC	—	50	M,C	F,N	16	120	82S226
82S129	256X4	TS	—	50	M,C	F,N	16	120	82S229
10149	256X4	OE	—	20	C	F	16	150	—
82S114	256X8	TS	—	60	M,C	F,N	24	175	82S214
82S130	512X4	OC	—	50	M,C	F,N	16	140	82S230
82S131	512X4	TS	—	50	M,C	F,N	16	140	82S231
82S115	512X8	TS	—	60	M,C	F,N	24	175	82S215
82S140	512X8	OC	—	60	M,C	F,N	24	175	82S240
82S141	512X8	TS	—	60	M,C	F,N	24	175	82S241
82S136	1024X4	OC	—	60	M,C	F,N	18	140	—
82S137	1024X4	TS	—	60	M,C	F,N	18	140	—
82S180	1024X8	OC	—	70	M,C	F,N	24	175	82S280
82S181	1024X8	TS	—	70	M,C	F,N	24	175	82S281
82S2708	1024X8	TS	—	70	M,C	F,N	24	175	—
82S184	2048X4	OC	—	100	M,C	I	18	120	—
82S185	2048X4	TS	—	100	M,C	I	18	120	—
82S190	2048X8	OC	—	70	M,C	I	24	175	82S290
82S191	2048X8	TS	—	70	M,C	I	24	175	82S291
FPLAS									
82S100	16X48X8	TS	—	50	M,C	I,N	28	170	—
82S101	16X48X8	OC	—	50	M,C	I,N	28	170	—
PLAS									
82S200	16X48X8	TS	—	50	M,C	I,N	28	170	—
82S201	16X48X8	OC	—	50	M,C	I,N	28	170	—
FPGAS									
82S102	16X9	OC	—	30	M,C	I,N	28	170	—
82S103	16X9	TS	—	30	M,C	I,N	28	170	—

*To be announced

NOTES

1. Output circuit:
   OE = Open emitter
   OC = Open collector
   TS = Tri-state
2. Output logic:
   T = Transparent—input data appears on output during Write
   B = Blanked—output is blanked during Write
3. Temperature range:
   C = Commercial (0° C to +75° C)
   M = Military (-55° C to +125° C)
   All ECL 10,000 series (-30° C to +85° C)
4. Commercial (0° C to +75° C)

DESCRIPTION

The 82S114 and 82S115 are field programmable and include on-chip decoding and 2 chip enable inputs for ease of memory expansion. They feature tri-state outputs for optimization of word expansion in bused organizations. A D-type latch is used to enable the tri-state output drivers. In the Transparent Read mode, stored data is addressed by applying a binary code to the address inputs while holding Strobe high. In this mode the bit drivers will be controlled solely by CE1 and CE2 lines.

In the Latched Read mode, outputs are held in their previous state (high, low, or high Z) as long as Strobe is low, regardless of the state of address or chip enable. A positive Strobe transition causes data from the applied address to reach the outputs if the chip is enabled, and causes outputs to go to the high Z state if the chip is disabled.

A negative Strobe transition causes outputs to be locked into their last Read Data condition if the chip was enabled, or causes outputs to be locked into the high Z condition if the chip was disabled.

Both 82S114 and 82S115 devices are available in the commercial and military temperature ranges. For the commercial temperature range (0°C to +75°C) specify N82S114/115, F or N, and for the military temperature range (-55°C to +125°C) specify S82S114/115, F.

FEATURES

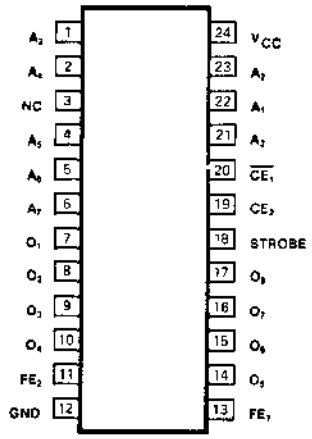
- **Address access time:**
  N82S114/115: 60ns max
  S82S114/115: 90ns max
- **Power dissipation: 165μW/bit typ**
- **Input loading:**
  N82S114/115: -100μA max
  S82S114/115: -150μA max
- **On-chip storage latches**
- **Schottky clamped**
- **Fully compatible with Signetics 82S214 and 82S215 ROMs**
- **Fully TTL compatible**

APPLICATIONS

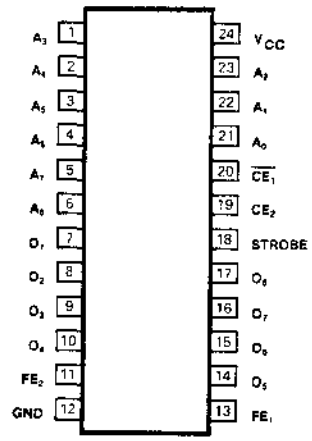
- **Microprogramming**
- **Hardwire algorithms**
- **Character generation**
- **Control store**
- **Sequential controllers**

PIN CONFIGURATIONS

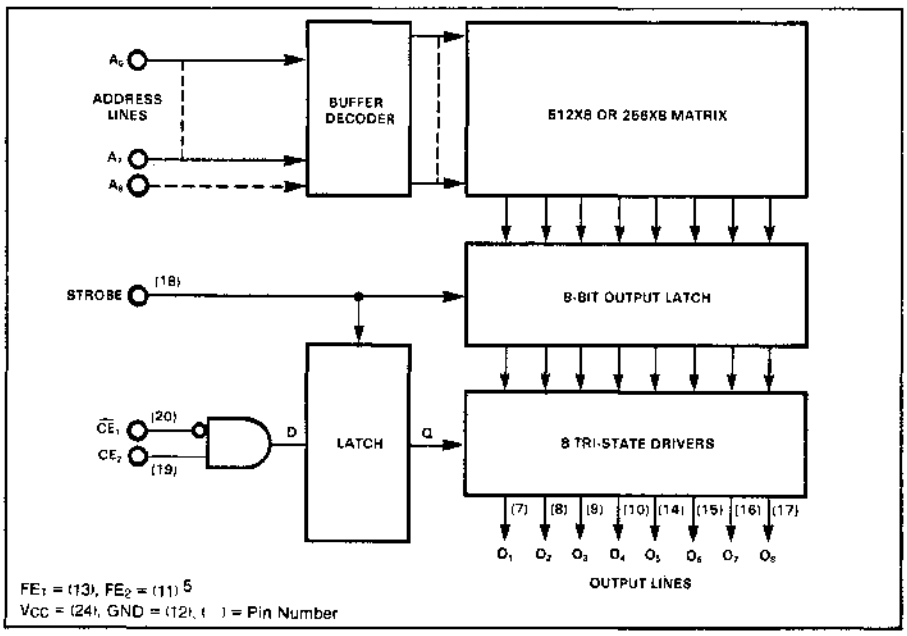
F,N PACKAGE
82S114



F,N PACKAGE
82S115



BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

PARAMETER		RATING	UNIT
V_{CC}	Supply voltage	+7	Vdc
V_{IN}	Input voltage	+5.5	Vdc
	Temperature range		°C
T_A	Operating		
	N82S114/115	0 to +75	
	S82S114/115	-55 to +125	
T_{STG}	Storage	-65 to +150	

DC ELECTRICAL CHARACTERISTICS

N82S114/115: $0°C \le T_A \le +75°C$, $4.75V \le V_{CC} \le 5.25V$
S82S114/115: $-55°C \le T_A \le +125°C$, $4.5V \le V_{CC} \le 5.5V$

PARAMETER		TEST CONDITIONS	N82S114/115			S82S114/115			UNIT
			Min	Typ[1]	Max	Min	Typ[1]	Max	
	Input voltage								V
V_{IL}	Low				.85			.8	
V_{IH}	High		2.0			2.0			
V_{IC}	Clamp	I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	
	Output voltage								V
V_{OL}	Low	I_{OUT} = 9.6mA		0.4	0.45		0.4	0.5	
V_{OH}	High	$\overline{CE}_1$ = Low, CE_2 = High, I_{OUT} = -2mA, High stored	2.7	3.3		2.4	3.3		
	Input current								μA
I_{IL}	Low	V_{IN} = 0.45V			-100			-150	
I_{IH}	High	V_{IN} = 5.5V			25			50	
	Output current								
$I_{O(OFF)}$	Hi-Z state	$\overline{CE}_1$ = High or CE_2 = O, V_{OUT} = 5.5V			40			100	μA
		$\overline{CE}_1$ = High or CE_2 = O, V_{OUT} = 0.5V			-40			-100	
I_{OS}	Short circuit[2]	V_{OUT} = OV	-20		-70	-15		-85	mA
I_{CC}	V_{CC} supply current			130	175		130	185	mA
	Capacitance	V_{CC} = 5.0V, V_{IN} = 2.0V							pF
C_{IN}	Input	V_{CC} = 5.0V, V_{OUT} = 2.0V		5			5		
C_{OUT}	Output	$\overline{CE}_1$ = High or CE_2 = 0		8			8		

AC ELECTRICAL CHARACTERISTICS

$R_1 = 470\Omega$, $R_2 = 1k\Omega$, $C_L = 30pF$
N82S114/115: $0° \le T_A \le +75°C$, $4.75V \le V_{CC} \le 5.25V$
S82S114/115: $-55°C \le T_A \le +125°C$, $4.5V \le V_{CC} \le 5.5V$

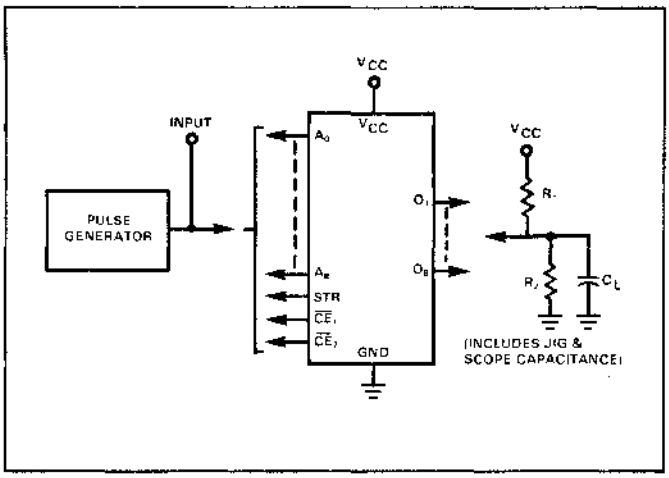
PARAMETER		TO	FROM	TEST CONDITIONS	N82S114/115			S82S114/115			UNIT
					Min	Typ[1]	Max	Min	Typ[1]	Max	
	Access time[3]			Latched or transparent read							ns
T_{AA}		Output	Address			35	60		35	90	
T_{CE}		Output	Chip enable			20	40		20	50	
	Disable time[3]			Latched or transparent read							ns
T_{CD}		Output	Chip disable			20	40		20	50	
	Setup and hold time[4]			Latched read only							ns
T_{CDS}	Setup time	Output	Chip		40			50			
T_{CDH}	Hold time		enable		10	0		10	0		
T_{ADH}	Hold time	Output	Address		0	-10		5	-10		
	Pulse width[4]			Latched read only							ns
T_{SW}	Strobe				30	20		40	20		
	Latch time[4]			Latched read only							ns
T_{SL}	Strobe				60	35		90	35		
	Delatch time[4]			Latched read only							ns
T_{DL}	Strobe						30			35	

NOTES on following page.

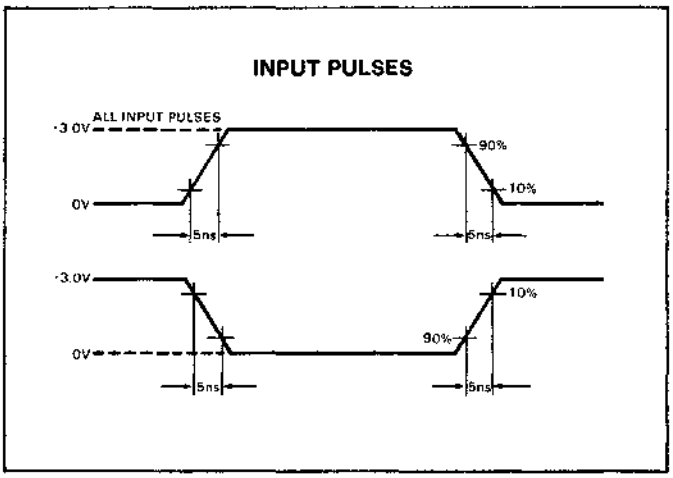
NOTES

1. Typical values are at V_{CC} = +5.0V and T_A = +25°C.
2. No more than one output should be grounded at the same time and strobe should be disabled. Strobe is in high state.
3. If the strobe is high, the device functions in a manner identical to conventional bipolar ROMs. The timing diagram shows valid data will appear T_A nanoseconds after the address has changed the T_{CE} nanoseconds after the output circuit is enabled. T_{CD} is the time required to disable the output and switch it to an off or high impedance state after it has been enabled.
4. In latched Read Mode data from any selected address will be held on the output when strobe is lowered. Only when strobe is raised will new location data be transferred and chip enable conditions be stored. The new data will appear on the outputs if the chip enable conditions enable the outputs.
5. During operation the fusing pins FE1 and FE2 may be grounded or left floating.
6. Positive current is defined as into the terminal referenced.

TEST LOAD CIRCUIT

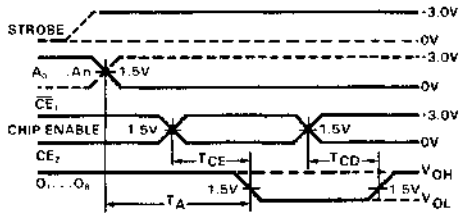


VOLTAGE WAVEFORM



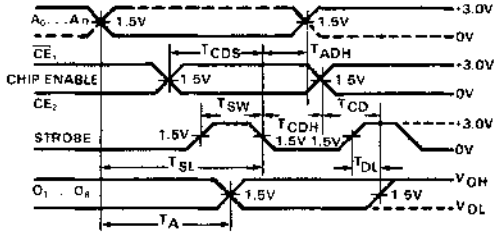
TIMING DIAGRAMS

TRANSPARENT READ



Output Latches Not Used

LATCHED READ



Output Latches Used

PROGRAMMING SYSTEMS SPECIFICATIONS (Testing of these limits may cause programming of device.) T_A = +25°C

PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
V_{CCP}	Power supply voltage To program[1]	I_{CCP} = 200 ± 25mA, Transient or steady state	4.75	5.0	5.25	V
	Verify limit					V
V_{CCH}	Upper		5.3	5.5	5.7	
V_{CCL}	Lower		4.3	4.5	4.7	
V_S	Verify threshold[2]		0.9	1.0	1.1	V
I_{CCP}	Programming supply current	V_{CCP} = +5.0 ± .25V	175	200	225	mA
	Input voltage					V
V_{IL}	Low		0	0.4	0.8	
V_{IH}	High		2.4		5.5	
	Input current (FE_1 & FE_2 only)					
I_{IL}	Low	V_{IL} = +0.45V			-100	μA
I_{IH}	High	V_{IH} = +5.5V			10	mA
	Input current (except FE_1 & FE_2)					μA
I_{IL}	Low	V_{IL} = +0.45V			-100	
I_{IH}	High	V_{IH} = +5.5V			25	
V_{OUT}	Output programming voltage[3]	I_{OUT} = 200 ± 20mA, Transient or steady state	16.0	17.0	18.0	V
I_{OUT}	Output programming current	V_{OUT} = +17 ± 1V	180	200	220	mA
T_R	Output pulse rise time		10		50	μs
t_P	FE_2 programming pulse width		0.3	0.4	0.5	ms
T_D	Pulse sequence delay		10			μs
T_{PR}	Programming time	V_{CC} = V_{CCP}			12	sec
T_{PS}	Programming pause	V_{CC} = 0V	6			sec
$\frac{T_{PR}}{T_{PR}+T_{PS}}$	Programming duty cycle[4]				50	%

NOTES

1. Bypass V_{CC} to GND with a 0.01μF capacitor to reduce voltage spikes.
2. Vs is the sensing threshold of the PROM output voltage for a programmed bit. It normally constitutes the reference voltage applied to a comparator circuit to verify a successful fusing attempt.
3. Care should be taken to insure the 17 ± 1V output voltage is maintained during the entire fusing cycle.
4. Continuous fusing for an unlimited time is also allowed, provided that a 60% duty cycle is maintained. This may be accomplished by following each Program-Verify cycle with a Rest period (V_{CC} = 0V) of 3ms.

RECOMMENDED PROGRAMMING PROCEDURE

The 82S114/115 are shipped with all bits at logical low. To write logical high, proceed as follows:

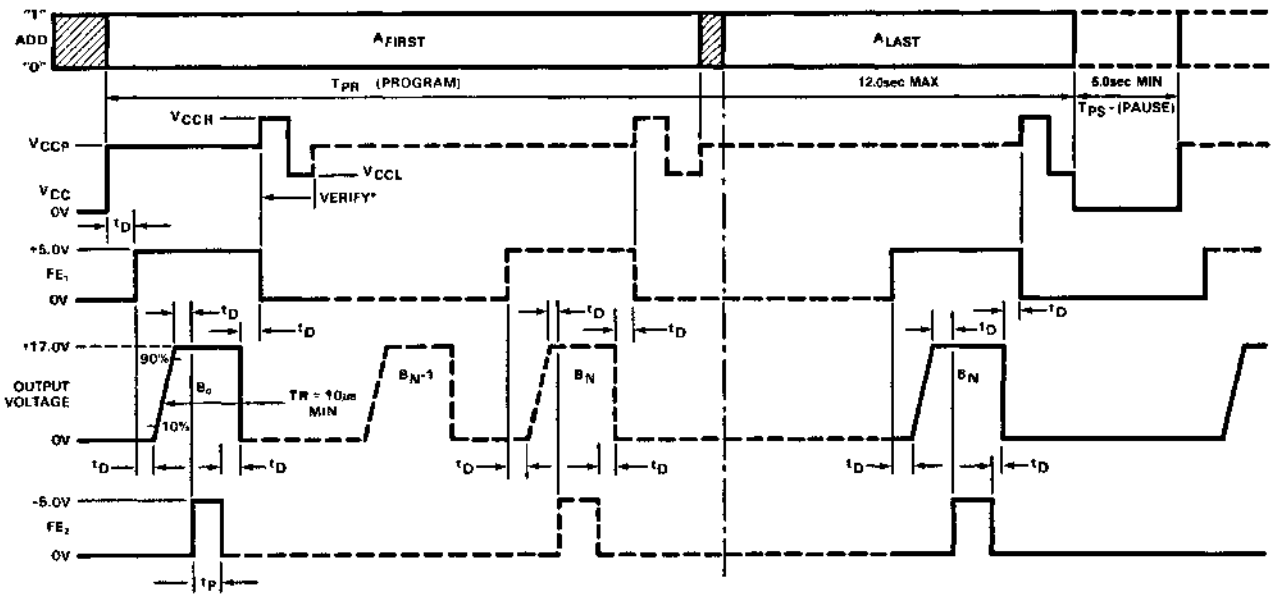
SET-UP

1. Apply GND to pin 12.
2. Terminate all device outputs with a 10kΩ resistor to V_{CC}.
3. Set $\overline{CE}_1$ to logic low, and CE_2 to logic high (TTL levels).
4. Set Strobe to logic high level.

Program-Verify Sequence

1. Raise V_{CC} to V_{CCP}, and address the word to be programmed by applying TTL high and low logic levels to the device address inputs.
2. After 10μs delay, apply to FE_1 (pin 13) a voltage source of +5.0 ± 0.5V, with 10mA sourcing current capability.
3. After 10μs delay, apply a voltage source of +17.0 ± 1.0V to the output to be programmed. The source must have a current limit 200mA. Program on output at the time.
4. After 10μs delay, raise FE_2 (pin 11) from 0V to +5.0 ± 0.5V for a period of 1ms, and then return to 0V. Pulse source must have a 10mA sourcing current capability.
5. After 10μs delay, remove +17.0V supply from programmed output.
6. To verify programming, after 10μs delay, return FE_1 to 0V. Raise V_{CC} to V_{CCH} = +5.5 ± .2V. The programmed output should remain in the high state. Again, lower V_{CC} to V_{CCL} = +4.5 ± .2V, and verify that the programmed output remains in the high state.
7. Raise V_{CC} to V_{CCP} and repeat steps 2 through 6 to program other bits at the same address.
8. Repeat steps 1 through 7 to program all other address locations.

TYPICAL PROGRAMMING SEQUENCE



*Programming verification at both high and low V_{CC} margins is optional. For convenience, verification can also be executed at the operating V_{CC} limits specified in the dc characteristics.

DESCRIPTION

The 82S16/116 and 82S17/117 are read/write memory arrays which feature either open collector or tri-state output options for optimization of word expansion in bused organizations. Memory expansion is further enhanced by full on-chip address decoding, 3 chip enable inputs and pnp input transistors which reduce input loading to 25μA for a high level, and −100μA for a low level.

During Write operation, the logical state of the output of both devices follows the complement of the data input being written. This feature allows faster execution of Write-Read cycles, enhancing the performance of systems utilizing indirect addressing modes, and/or requiring immediate verification following a Write cycle.

Both devices have fast read access and write cycle times, and thus are ideally suited in high-speed memory applications such as cache, buffers, scratch pads, writable control stores, etc.

All devices are available in the commercial temperature range (0°C to +75°C) and are specified as N82S16/116/17/117, F or N. The 82S16 and 82S17 are also available in the military temperature range (−55°C to +125°C) and are specified as S82S16/17.

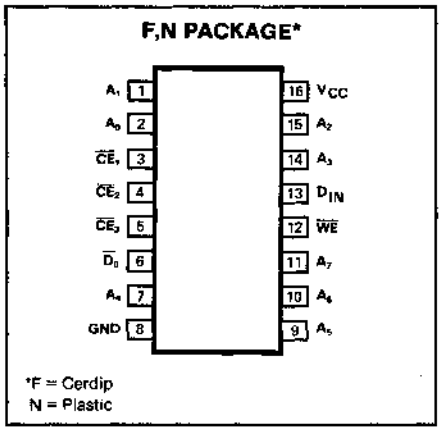
FEATURES

- **Address access time:**
  82S116/117: 40ns max
- **Write cycle time:**
  82S116/117: 25ns max
- **Power dissipation: 1.5mW/bit typ**
- **Input loading:**
  N82S116/117: −100μA
- **Output follows complement of data input during Write**
- **On-chip address decoding**
- **Output option:**
  82S16/116: Tri-state
  82S17/117: Open collector
- **Schottky clamped**
- **TTL compatible**

APPLICATIONS

- **Buffer memory**
- **Writable control store**
- **Memory mapping**
- **Push down stack**
- **Scratch pad**

PIN CONFIGURATION

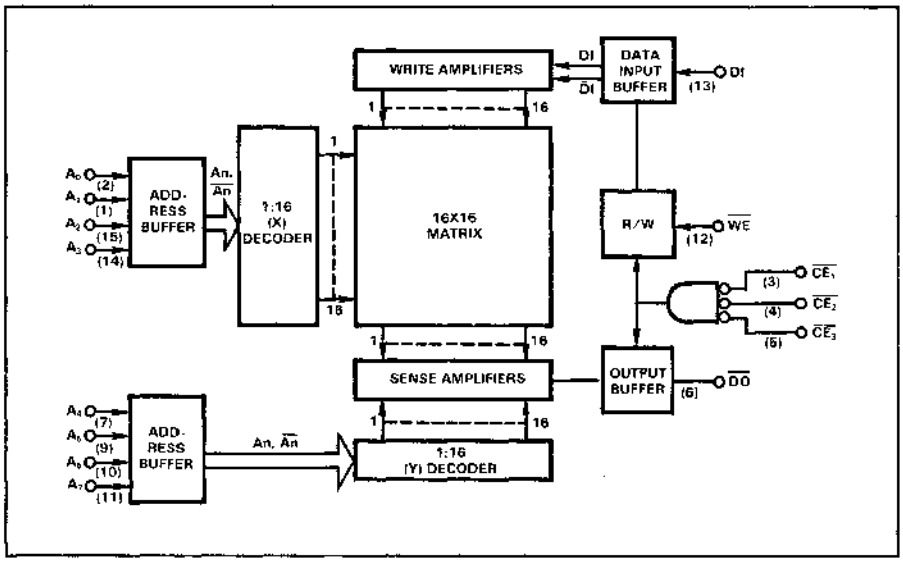


TRUTH TABLE

MODE	$\overline{CE}$*	$\overline{WE}$	D_{IN}	$\overline{D}_{OUT}$	
				82S16/116	32S17/117
Read	0	1	X	Stored $\overline{data}$	Stored $\overline{data}$
Write "0"	0	0	0	1	1
Write "1"	0	0	1	0	0
Diasabled	1	X	X	High-Z	1

*"0" = All $\overline{CE}$ inputs low; "1" = one or more $\overline{CE}$ inputs high.
X = Don't care.

BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

PARAMETER		RATING	UNIT
V_{CC}	Supply voltage	+7	Vdc
V_{IN}	Input voltage	+5.5	Vdc
	Output voltage	+5.5	Vdc
V_{OUT}	High (82S17)		
V_O	Off-state (82S16)		
	Temperature range		°C
T_A	Operating		
	S82S16/17	−55 to + 125	
	N82S16/17, N82S116/117	0 to +75	
T_{STG}	Storage	−65 to +150	

DC ELECTRICAL CHARACTERISTICS

N82S116/117, N82S16/17: 0°C ≤ T_A ≤ +75°C, 4.75V ≤ V_{CC} ≤ 5.25V

S82S16/17: −55°C ≤ T_A ≤ +125°C, 4.5V ≤ V_{CC} ≤ 5.5V

PARAMETER		TEST CONDITIONS	N82S16/17/116/117			S82S16/17			UNIT
			Min	Typ[1]	Max	Min	Typ[1]	Max	
	Input voltage[2]								V
V_{IH}	High	V_{CC} = Max	2.0			2.0			
V_{IL}	Low	V_{CC} = Min			0.85			0.8	
V_{IC}	Clamp[3]	V_{CC} = Min, I_{IN} = −12mA		−1.0	−1.5		−1.0	−1.5	
	Output voltage[2]	V_{CC}= Min							V
V_{OH}	High (82S16/116)[4]	I_{OH} = −3.2mA	2.6			2.4			
V_{OL}	Low[5]	I_{OL} = 16mA		0.35	0.45		0.35	0.5	
	Input current[3]	V_{CC} = Max							mA
I_{IH}	High	V_{IN} = 5.5V		1	25		1	25	
I_{IL}	Low	V_{IN} = 0.45V		−10	−100		−10	−250	
	Output current								
I_{OLK}	Leakage (82S17/117)[6]	V_{OUT} = 5.5V		1	40		1	40	μA
$I_{O(OFF)}$	Hi-Z state (82S16/116)[6]	V_{OUT} = 5.5V		1	40		1	50	μA
		V_{OUT} = 0.45V		−1	−40		−1	−50	
I_{OS}	Short-circuit (82S16/116)[7]	V_{CC} = Max, V_O = 0V	−20		−70	−20		−70	μA
I_{CC}	V_{CC} supply current[8]	V_{CC} = Max		80	115		80	120	mA
	Capacitance	V_{CC} = 5.0V							pF
C_{IN}	Input	V_{IN} = 2.0V		5			5		
C_{OUT}	Output	V_{OUT} = 2.0V		8			8		

AC ELECTRICAL CHARACTERISTICS

$R_1 = 270\Omega$, $R_2 = 600\Omega$, $C_L = 30pF$

N82S116/117, N82S16/17: $0°C \le T_A \le +75°C$, $4.75V \le V_{CC} \le 5.25V$

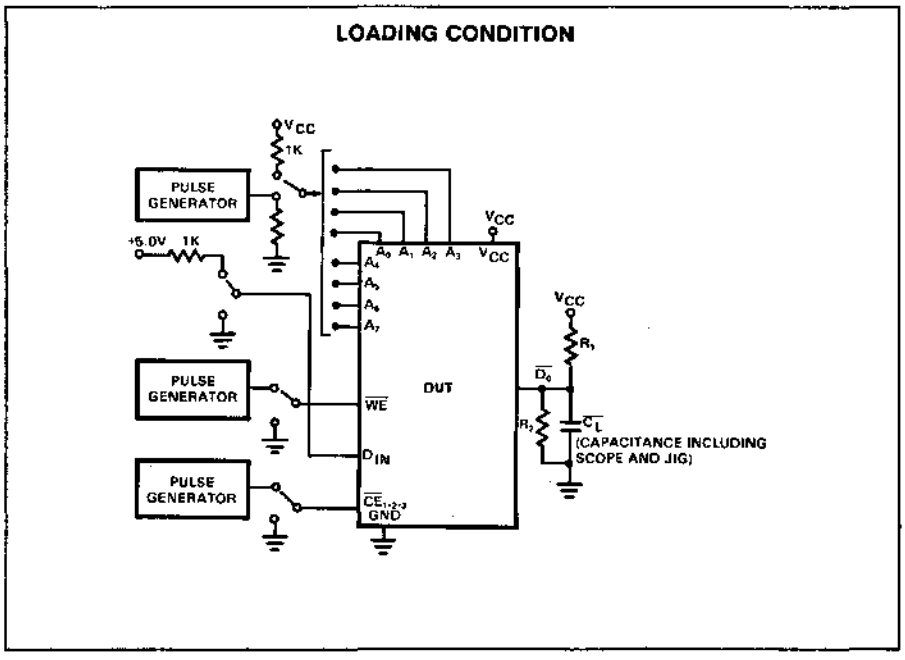
S82S16/17: $-55°C \le T_A \le +125°C$, $4.5V \le V_{CC} \le 5.5V$

PARAMETER		TO	FROM	N82S16/17			N82S116/117			S82S16/17			UNIT
				Min	Typ[1]	Max	Min	Typ[1]	Max	Min	Typ[1]	Max	
	Access time												ns
T_{AA}	Address				40	50		30	40		40	70	
T_{CE}	Chip enable				30	40		15	25		30	40	
T_{CD}	Disable time	Output	Chip enable		30	40		15	25		30	40	ns
T_{WD}	Valid time	Output	Write enable		30	40		30	40		30	55	ns
	Setup and hold time												ns
T_{WSA}	Setup time	Write enable	Address	20	5		0	-5		20	5		
T_{WHA}	Hold time			5	0		0	-5		10	0		
T_{WSD}	Setup time	Write enable	Data in	40	30		25	15		50	40		
T_{WHD}	Hold time			5	0		0	-5		10	0		
T_{WSC}	Setup time	Write enable	$\overline{CE}$	10	0		0	-5		10	0		
T_{WHC}	Hold time			5	0		0	-5		10	0		
	Pulse width												ns
T_{WP}	Write enable[9]			30	15		25	15		40	20		

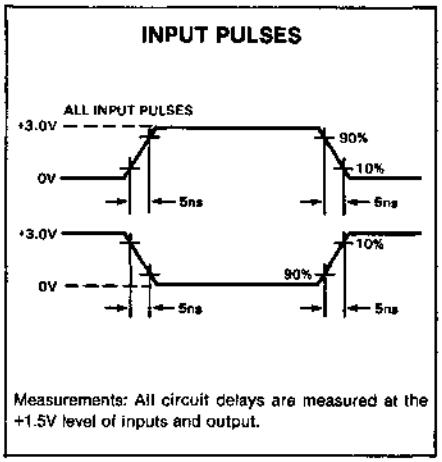
NOTES

1. All typical values are at V_{CC} = 5V, T_A +25°C.
2. All voltage values are with respect to network ground terminal.
3. Test each input one at the time.
4. Measured with a logic low stored and V_{IL} applied to $\overline{CE_1}$, $\overline{CE_2}$ and $\overline{CE_3}$.
5. Measured with a logic high stored. Output sink current is supplied through a resistor to V_{CC}.
6. Measured with V_{IH} applied to $\overline{CE1}$, $\overline{CE2}$ and $\overline{CE3}$.
7. Duration of the short-circuit should not exceed 1 second.
8. I_{CC} is measured with the write enable and memory enable inputs grounded, all other inputs at 4.5V, and the output open.
9. Minimum required to guarantee a Write into the slowest bit.

TEST LOAD CIRCUIT



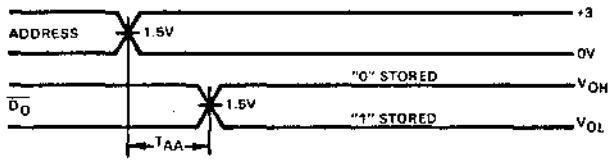
VOLTAGE WAVEFORM



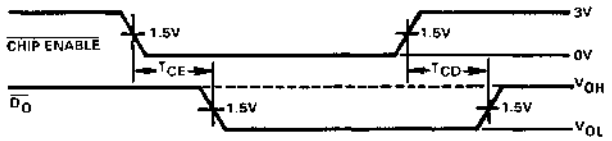
Measurements: All circuit delays are measured at the +1.5V level of inputs and output.

TIMING DIAGRAMS

READ CYCLE

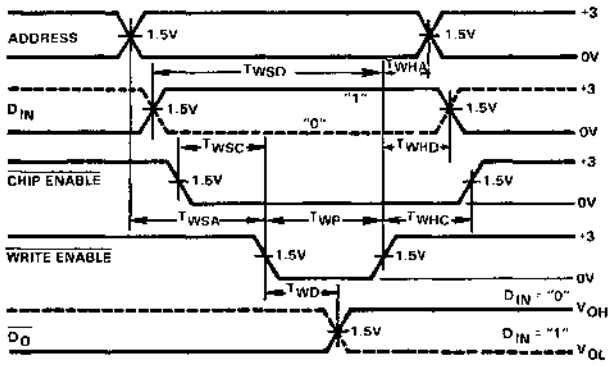


Address Access Time



Chip Enable/Disable Times

WRITE CYCLE



MEMORY TIMING DEFINITIONS

T_{CE}	Delay between beginning of Chip Enable low (with Address valid) and when Data Output becomes valid.
T_{CD}	Delay between when Chip Enable becomes high and Data Output is in off state.
T_{AA}	Delay between beginning of valid Address (with Chip Enable low) and when Data Output becomes valid.
T_{WSC}	Required delay between beginning of valid Chip Enable and beginning of Write Enable pulse.
T_{WHD}	Required delay between end of Write Enable pulse and end of valid Input Data.
T_{WP}	Width of Write Enable pulse.
T_{WSA}	Required delay between beginning of valid Address and beginning of Write Enable pulse.
T_{WSD}	Required delay between beginning of valid Data Input and end of Write Enable pulse.
T_{WD}	Delay between beginning of Write Enable pulse and when Data Output reflects complement of Data Input.
T_{WHC}	Required delay between end of Write Enable pulse and end of Chip Enable.
T_{WHA}	Required delay between end of Write Enable pulse and end of valid Address.

APPENDIX

DESCRIPTION

The Signetics 8X300 Fixed Instruction Bipolar Microprocessor is a high performance microprocessor optimized for control and data movement applications.

The unique features of the 8X300 IV bus and instruction set permit 8-bit parallel data to be rotated or masked before undergoing arithmetic or logical operations. Then, the data may be shifted and merged into any field of from 1 to 8 contiguous bits at the destination. The entire process of input, shifting, processing and output is accomplished in one instruction cycle time. The 250ns cycle time makes the 8X300 suited for high speed applications.

The evaluation board contains all the elements which a designer needs to familiarize himself with the 8X300 for his systems applications. Included with the 8X300 are 4 I/O ports for external device interface, 256 bytes of temporary (working) data storage and 512 words of program storage, all properly connected to the 8X300 to allow immediate exercising of the board. For this purpose, the PROMs are preprogrammed with the I/O control, RAM control and RAM integrity diagnostic programs. With the remaining PROM space, the designer may enter his own benchmark, test or development routines.

The board design allows complete flexibility in access to the address, instruction and IV busses as well as all controls and signals of the 8X300. The IV bus, I/O port user connection, clock signals, control lines, address bus and instruction bus are wired to output pins, the board edge connector and flat cable connectors.

The board layout permits variations and/or expansion of the basic design. In addition to the access to all signals for transfer off the board, a wire wrap area is provided so that the designer may add to the board circuitry as he desires. The addition may include memory, additional interfaces, or special circuits which meet specific user requirements.

Controls are also provided for diagnostic and instructional purposes by allowing various operating modes. In the WAIT mode, the program may be single stepped for ease of checkout. The one-shot instruction jamming allows control of the program start location, changes of program flow, changing or examining the internal registers, or testing of simple sequences. The repeated instruction jamming provides a means of repetitive execution of an instruction so that the I/O bus and the control lines may be examined without software changes. In both of these jam cases, the jammed instruction is selected by board-mounted switches.

Auxiliary Circuits

The 8X300 can be used with any bipolar (or TTL-compatible) ICs. It can directly address 8192 program instruction locations and up to 512 I/O ports. Memory paging may be employed for larger working storage. Typical auxiliary circuits include:

DEVICE	MFG	PART NUMBER	DESCRIPTION	QTY
U14	Signetics	N8X300I	Microprocessor	1
U2-U9		N82S116B	RAM (256X1)	8
U15-16		N82S115I	PROM (512X8)	2
U13, U21, U26, U27		N8T32N	I/O Port (Tri-state)	4
U1		N8T31N	Latch	1
U10-U11		N8T26AB	Transceiver	2
U17-U20		N74LS157B	Multiplexer	4
U23, U25		N74LS74A	Flip-Flop	2
U12		N74LS27A	NOR Gate	1
U22, U24	Signetics	N74LS00A	NAND Gate	2
Q1		2N5320	Transistor	1
Q2		2N2222	Transistor	1
RN1-RN3	Dale	CSP-10E-01-102-K	Resistor network	3
R1, R3, R4		1k Ω 1/4W	Resistor	3
R5		4k Ω 1/4W	Resistor	1
R2		8k Ω 1/4W	Resistor	1
C2-C20, C22-C30		0.1μF	Capacitor	29
C1, C21		22μF	Capacitor	2
Z1		8.00 MHz	Crystal	1
J104	Berg	65616-134	Connector	1
J101	Berg	65616-150	Connector	1
J105	Berg	65616-120	Connector	4
	Robinson-Nugent	ICN-163-53	Socket	2
	Robinson-Nugent	ICN-246-54	Socket	7
	Robinson-Nugent	SB-25	Strip socket	2
S1, S2	Amp	435640-5	Dip switch	2
S5, S6	Alco	MSTS-104D	Toggle switch	2
S3, S4	Digitast	ST	Momentary switch	2
	Smith	230 (red)	Binding post	1
	Smith	230 (black)	Binding post	1
	SAE	CPH 8100-100	50/100 edge connector	1
			PC board	1
			Packing case	1
			Assembly manual	1

Table 1 KIT CONTENTS

Program Storage
- 82S115 (512X8 PROM)

I/O
- 8T32/33 (8-Bit Synchronous Bidirectional I/O Port)
- 8T35/36 (8-Bit Asynchronous Bidirectional I/O Port)
- 8T31 (8-Bit Bidirectional I/O Port)
- 8T39 (Quad Bus Extender)

Working Storage
- 82S116 (256X1 RAM)

KIT ASSEMBLY

The kit is designed to be assembled by a skilled technician. To aid assembly as well as the evaluation, the major board areas are identified and part placement is indicated directly on the PC board. The board has been solder masked so that it may be wave soldered and to avoid solder bridges if the board is hand soldered. Sockets are provided to mount the following parts: 8X300, 8T32 (4), 8T31, 82S115 (2), and the DIP Switches (2). Kit assembly is straightforward and may be accomplished in about four hours.

Figure 4 shows the component side of the board and the position and orientation of all parts. Assembly can be accomplished using this figure, the notes and observing the board markings. Figure 5 provides the schematic.

SOFTWARE

The PROMs furnished with the kit are programmed with diagnostics in the first 50 locations to assist in verifying that the assembled kit is working correctly. The remaining 462 locations may be programmed by the user.

The diagnostics are separated into two parts: Locations 00-23 contain tests of the I/O ports and the interface to the RAM, locations 24-47 write all combinations of bits into all locations of RAM and test the data read back. Locations 50-61 contain a delay routine which is used by the memory test to allow monitoring of the test with an external LED array or logic analyzer.

Execution of the first part should be checked in Single Step. Figures 1, 2 and 3 are flow diagrams for the diagnostic coding.

CONTROLS

Run/Wait

With this switch in the WAIT position, the processor halts execution of instructions and holds all buses in their current state (MCLK will still be active). When the switch is returned to the RUN position, the processor will remain in the halted state until the STEP key is depressed. The processor then begins running normally and subsequent depressions of the STEP key have no effect.

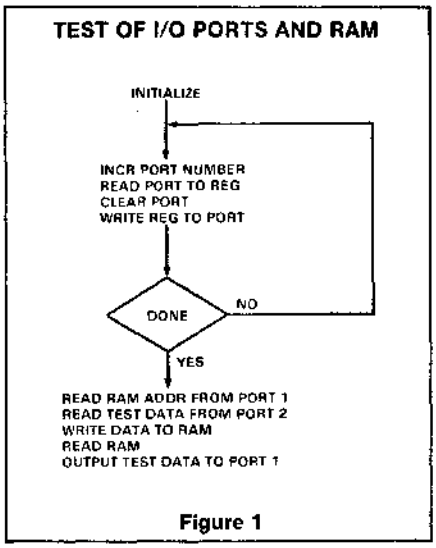


Figure 1

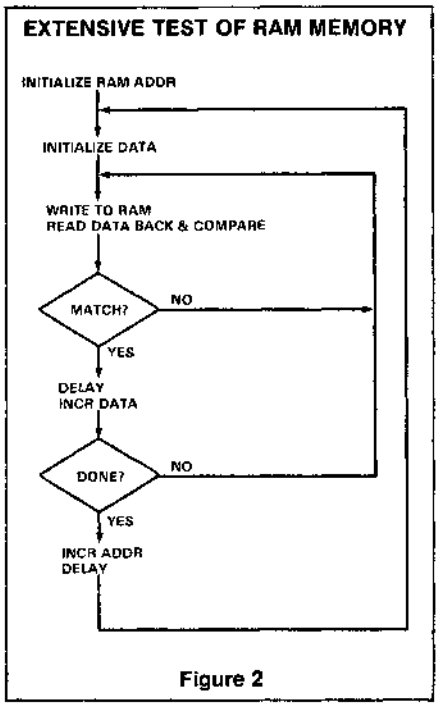


Figure 2

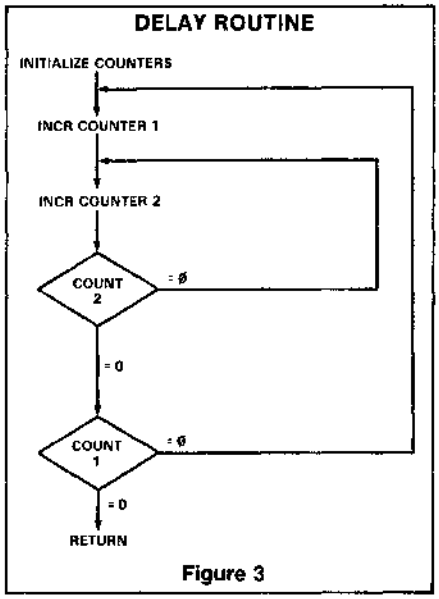


Figure 3

Single Step

Instructions may be executed one at a time by depressing the STEP key while the processor is halted. Instructions can be read from either the PROM or the Instruction switches.

Instruction Insertion

Although instructions are normally fetched from the PROMs, the bank of DIP switches provides an alternate instruction source. The command that is set in the switches is jammed on the instruction bus for either 1 cycle or indefinitely, depending on the setting on the SINGLE/REPEAT switch.

With the switch in the SINGLE position, the instruction encoded in the switches is placed on the bus for 1 cycle time when the INSERT key is pressed, then control returns to the PROMs. With the switch in the REPEAT position, the instruction will be executed repeatedly until the switch is returned to the SINGLE position. This feature allows examination of the control signals without changing the software.

Connection to the I/O Ports

The I/O ports are 8T32s programmed with addresses 1 through 4. It is recommended that $\overline{BOC}$ (J105 pin 18) be tied low for most tests and that $\overline{BIC}$ (J105 pin 20) be pulled low when data is to be entered to the port from an external device. This scheme allows output from the port to be monitored constantly yet data can be entered at any time since $\overline{BIC}$ overrides $\overline{BOC}$ (see 8T32 data sheet).

USAGE

By external means, a test pattern is loaded into each of the I/O ports. As the program is stepped through, each of the ports is read, cleared, then restored to verify proper connection of the control and data buses. If the connections are properly made, the test pattern that the user entered into Port 1 will be cleared after 10 steps and restored on the 11th. The other 3 ports are tested in sequence after 6 more steps per port. The memory is tested in a similar manner for continuity of the control signals by reading an address from Port 1 and a test pattern from Port 2, writing the pattern to the addressed location, reading the data back and writing it to Port 1. If the data in Port 1 matches the data in Port 2, the control interface to the RAM is verified. The flow diagram for this test is given in Figure 1.

The next portion of the test exercises each location in the RAM with all possible combination of bits. If the pattern read back does not match the data written, the test will be repeated with the same address and pattern until successful. The RAM address being written is output to Port 1, the pattern to Port 2. S5 should be in the RUN mode for this test. The program includes a delay loop so that its progress may be monitored by a simple LED display connected to the 8T32 outputs (Ports 1 and 2).

COMPONENT SIDE OF 8X300 PC BOARD

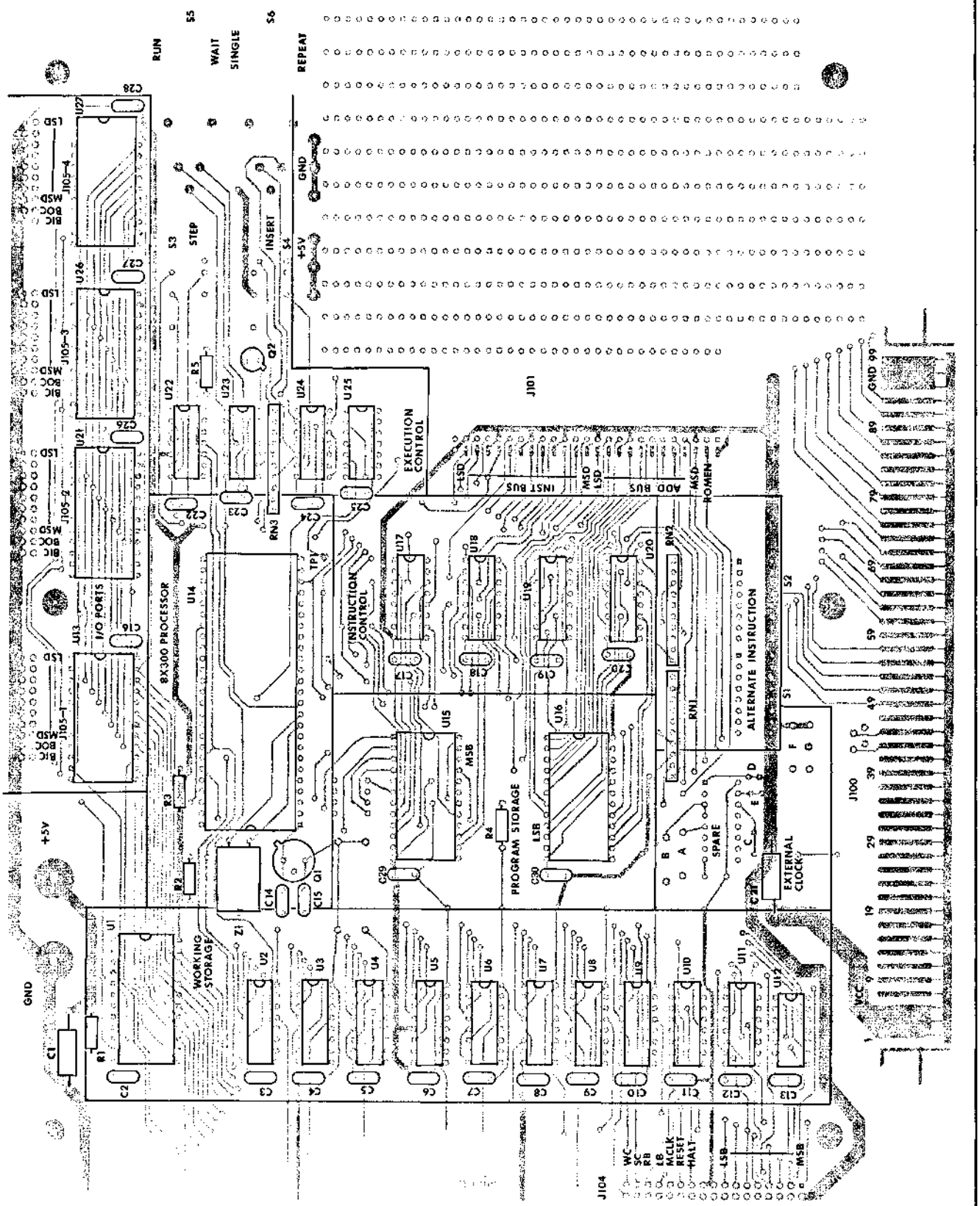


Figure 4

8X300 SCHEMATIC

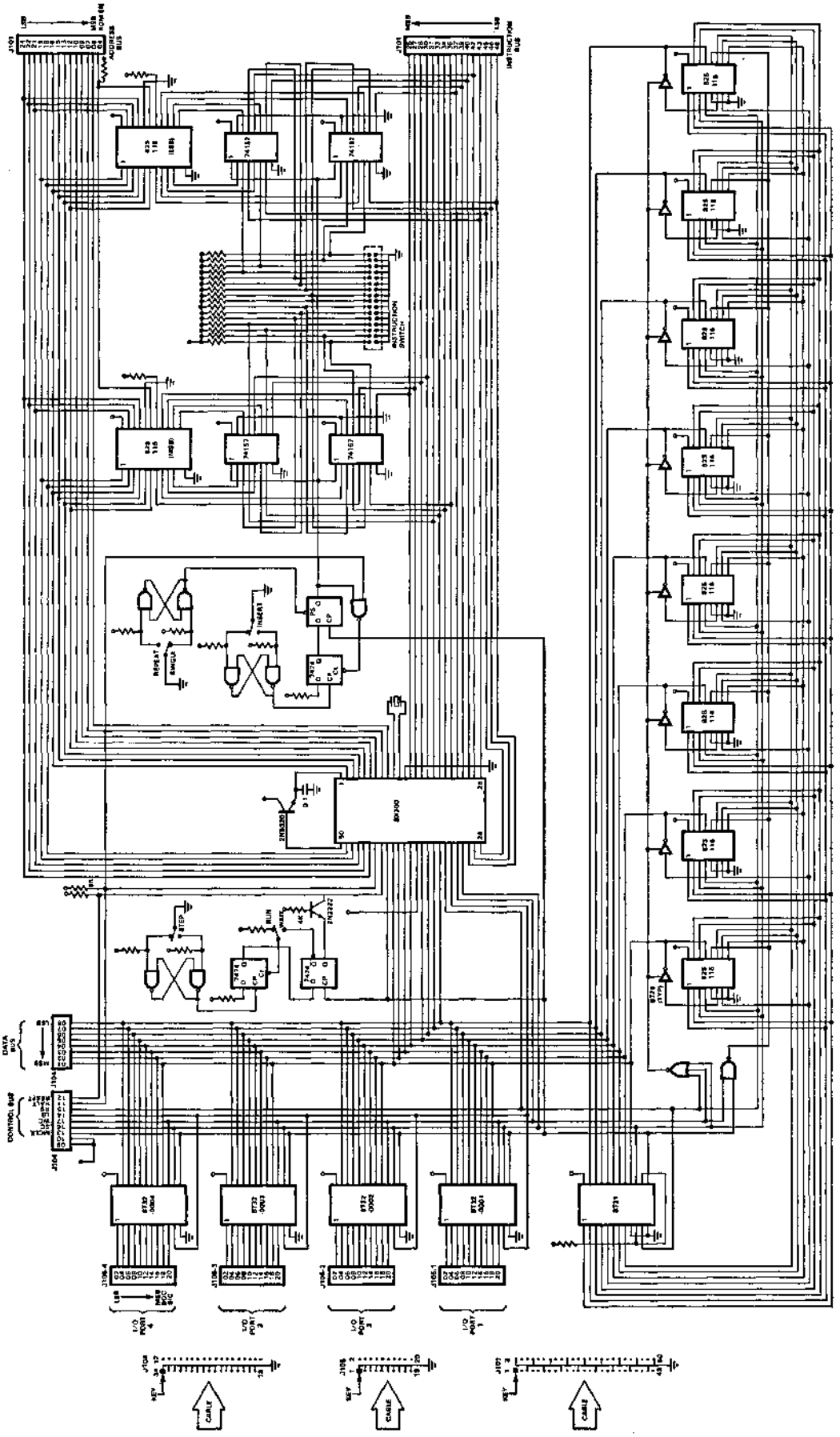
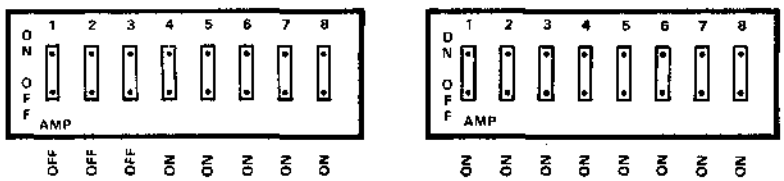


Figure 5

PIN NO.	FUNCTION		PIN NO.	FUNCTION	
1	VCC		2	VCC	
3	VCC		4	VCC	
5	VCC		6	VCC	
7	———		8	D07 (MSB)	DATA BUS
9	———		10	D06	
11	———		12	D05	
13	———		14	D04	
15	———		16	D03	
17	———		18	D02	
19	———		20	D01	
21	———		22	D00 (LSB)	
23	———		24	RESET	CONTROL BUS
25	———		26	HALT	
27	———		28	MCLK	
29	———		30	LB	
31	———		32	RB	
33	———		34	SC	
35	———		36	WC	
37	GND		38	GND	
39	———		40	CLK OUT	
41	———		42	CLK IN	
43	SPARE		44	SPARE	
45	SPARE		46	SPARE	
47	GND		48	GND	
49	I00	INSTRUCTION BUS	50	———	
51	I01		52	———	
53	GND		54	GND	
55	I04		56	A00	ADDRESS BUS
57	I05		58	A01	
59	GND		60	GND	
61	I08		62	A04	
63	I09		64	A05	
65	GND		66	GND	
67	I12		68	A08	
69	I13		70	A09	
71	GND		72	GND	
73	I15		74	A12	
75	I14		76	ROM EN	
77	GND		78	GND	
79	I11		80	A11	
81	I10		82	A10	
83	GND		84	GND	
85	I07		86	A07	
87	I06		88	A06	
89	GND		90	GND	
91	I03		92	A03	
93	I02		94	A02	
95	GND		96	GND	
97	GND		98	GND	
99	GND		100	GND	

Table 2 J100 CONNECTIONS

JUMP TO TEST START SELECTION (700000_8)



Alternate Instruction DIP Switches

Figure 6

To begin the test sequence, the DIP switches are loaded with 700000_8 (a jump to address 0), as shown in Figure 6. Switches S5 and S6 are placed in the WAIT and SINGLE positions respectively. After power is applied to the board, S4 must be pressed to insert the jump to 0. The system is now ready to be stepped through the diagnostic programs. A listing of the diagnostic coding is given in Table 3.

External Clock Synchronization

The 8X300 board provides for clock synchronization with external logic by means of the "spare" IC location at the lower center part of the board.

To drive the 8X300 from an external TTL level clock source, install an 8T98 driver in the "spare" location, place 100 Ω resistors at points A and B and 47 Ω resistors at points F and G. Trace C may be cut to reduce the loading on MCLK is desired. Signal is input on J100 pin 42.

To synchronize external logic to the 8X300 with MCLK, install an 8T98 driver in the "spare" location. No traces need to be cut or jumpered. Signal is available on J100 pin 40.

Use with MCSIM*

Remove 8X300 from PC board and plug MCSIM cable into J104. Set on-board controls to RUN. When power is applied, MCSIM is ready to run according to the usual MCSIM procedures.

Use with SMS Monitor*

The 8X300 remains in the PC board when the SMS monitor is used. The panel connects to J104 and J101. Set the on-board controls to RUN and SINGLE, the procedure according to the monitor operating instructions. If the board is halted, depress the STEP switch to initiate.

JUMPERS FOR 82S114/82S115 PROMs

The 8X300 Evaluation Board will accommodate either 82S114 or 82S115 PROMs. Determine which memory was included in this kit and connect the circuit according to the appropriate drawing in Figure 7. Use insulated wires approximately 0.5" long with all but 0.3" insulation stripped off. Put wires in place, solder and trim.

*Scientific Microsystems, 520 Clyde Avenue, Mt. View, Calif. 94043

ADDRESS	CODE	INSTRUCTION		COMMENT
TEST OF I/O PORTS & RAM				
000	6 06374		XMIT -4, R6	Initialize Counter
001	6 11000		XMIT 0, R11	Initialize Port Register
002	6 00001		XMIT 1, AUX	Load Increment
003	1 06006	Q1	ADD R6, R6	Incr Count
004	1 11011		ADD R11, R11	Incr Port Number
005	0 11007		MOVE R11, IVL	Select Port
006	0 27001		MOVE LB, R1	Read Port
007	6 27000		XMIT 0, LB	Clear Port
010	0 01027		MOVE R1, LB	Write Port
011	5 06003		NZT R6, Q1	All Done ?
012	6 07001		XMIT 1, IVL	Address Port 1
013	0 27001		MOVE LB, R1	Read RAM Addr
014	6 07002		XMIT 2, IVL	Addr Port 2
015	0 27002		MOVE LB, R2	Read RAM Data
016	0 01017		MOVE R1, IVR	Addr RAM
017	0 02037		MOVE R2, RB	Write RAM
020	6 02000		XMIT 0, R2	Clear R2
021	6 07001		XMIT 1, IVL	Addr Port 1
022	0 37002		MOVE RB, R2	Read RAM
023	0 02027		MOVE R2, LB	Output Data to Port 1
BEGINNING OF MEMORY TEST				
024	6 01000		XMIT 0, R1	Initialize RAM Addr
025	6 07001	Q2	XMIT 1, IVL	Addr Port 1
026	0 01027		MOVE R1, LB	Output RAM Addr
027	0 27017		MOVE LB, IVR	Addr RAM
030	6 02000		XMIT 0, R2	Initialize RAM Data
031	6 07002		XMIT 2, IVL	Addr Port 2
032	0 02027	Q3	MOVE R2, LB	Output RAM Data
033	0 02037		MOVE R2, RB	Write to RAM
034	0 02000		MOVE R2, AUX	Move Data to AUX
035	3 37000		XOR RB, AUX	Compare
036	5 00032		NZT AUX, Q3	Test
037	6 11000		XMIT 0, R11	Set up Return Indicator
040	7 00050		JMP DELAY	Delay for display purposes
041	6 00001		XMIT 1, AUX	Load Incr
042	1 02002		ADD R2, R2	Incr Data
043	5 02032		NZT R2, Q3	Done?
044	1 01001		ADD R1, R1	Incr Addr
045	6 11001		XMIT 1, R11	Set up Return Indicator
046	7 00050		JMP DELAY	Delay
047	7 00025		JMP Q2	Repeat
050	6 03000	DELAY	XMIT 0, R3	
051	6 04000		XMIT 0, R4	
052	6 00001		XMIT 1, Aux	
053	1 03003	D1	ADD R3, R3	
054	1 04004	D2	ADD R4, R4	
055	5 04054		NZT R4, D2	
056	5 03053		NZT R3, D1	
057	4 11060	RTN	XEC *+1, R11	
060	7 00041		JMP 041	
061	7 00047		JMP 047	

Table 3 SYSTEM TEST

JUMPERS FOR 82S114/115 PROMS

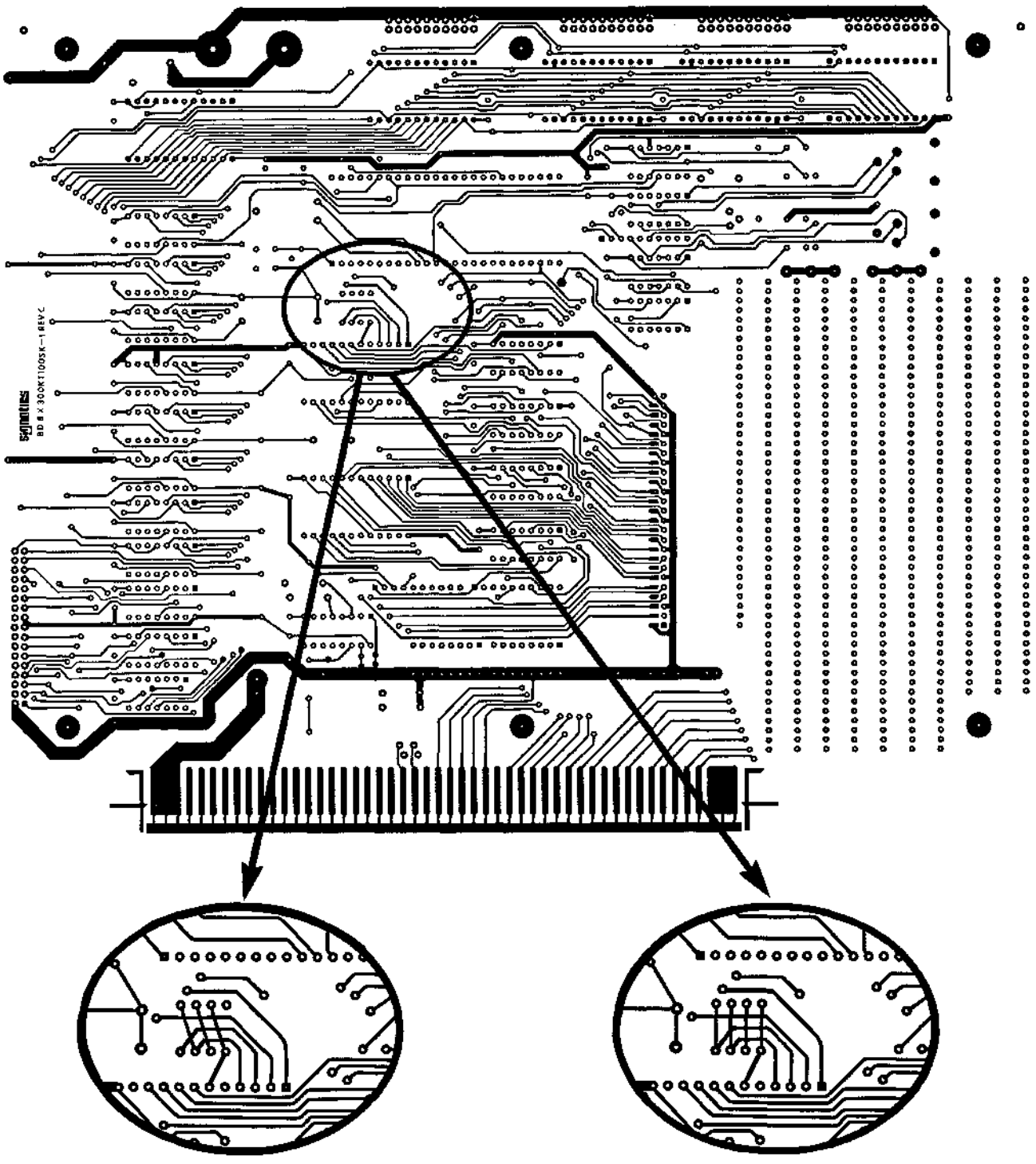


Figure 7

SALES OFFICES

SIGNETICS HEADQUARTERS

811 East Arques Avenue
Sunnyvale, California 94086
Phone: (408) 739-7700

ALABAMA
Huntsville
Phone: (205) 533-5800

ARIZONA
Phoenix
Phone: (602) 971-2517

CALIFORNIA
Inglewood
Phone: (213) 670-1101
Irvine
Phone: (714) 833-8980
(213) 924-1668
San Diego
Phone: (714) 560-0242
Sunnyvale
Phone: (408) 736-7565

COLORADO
Parker
Phone: (303) 841-3274

FLORIDA
Pompano Beach
Phone: (305) 782-8225

ILLINOIS
Rolling Meadows
Phone: (312) 259-8300

KANSAS
Wichita
Phone: (316) 683-5652

MASSACHUSETTS
Woburn
Phone: (617) 933-8450

MARYLAND
Columbia
Phone: (301) 730-8100

MINNESOTA
Edina
Phone: (612) 835-7455

NEW JERSEY
Cherry Hill
Phone: (609) 665-5071
Piscataway
Phone: (201) 981-0123

NEW YORK
Wappingers Falls
Phone: (914) 297-4074
Woodbury, L.I.
Phone: (516) 364-9100

OHIO
Worthington
Phone: (614) 888-7143

TEXAS
Dallas
Phone: (214) 661-1296

REPRESENTATIVES

ALABAMA
Huntsville
Murcota
Phone: (205) 539-8476

CALIFORNIA
San Diego
Mesa Engineering
Phone: (714) 278-8021
Sherman Oaks
Astralonics
Phone: (213) 990-5903

CANADA
Calgary, Alberta
Philips Electronics Industries Ltd.
Phone: (403) 543-5711
Montreal, Quebec
Philips Electronics Industries Ltd.
Phone: (514) 342-9180
Ottawa, Ontario
Philips Electronics Industries Ltd.
Phone: (613) 237-3131
Scarborough, Ontario
Philips Electronics Industries Ltd.
Phone: (416) 292-5161
Vancouver, B.C.
Philips Electronics Industries Ltd.
Phone: (604) 435-4411

COLORADO
Denver
Barnhill Five, Inc.
Phone: (303) 426-8222

CONNECTICUT
Newtown
Kanan Associates
Phone: (203) 426-8157

FLORIDA
Altamonte Springs
Semtronic Associates
Phone: (305) 831-8233
Clearwater
Semtronic Associates
Phone: (813) 461-4675

ILLINOIS
Chicago
L-Tec, Inc.
Phone: (312) 286-1500

INDIANA
Indianapolis
Enco Marketing
Phone: (317) 545-5511

KANSAS
Overland Park
Advanced Technology Sales
Phone: (913) 492-4333

MARYLAND
Glen Burnie
Microcomp, Inc.
Phone: (301) 247-0400

MASSACHUSETTS
Reading
Kanan Associates
Phone: (617) 944-8484

MICHIGAN
Bloomfield Hills
Enco Marketing
Phone: (313) 642-0203

MINNESOTA
Edina
Mel Foster Tech. Assoc.
Phone: (612) 835-2254

MISSOURI
St. Louis
Advanced Technology Sales
Phone: (314) 567-6272

NEW JERSEY
Haddonfield
Thomas Assoc., Inc.
Phone: (609) 854-3011

NEW MEXICO
Albuquerque
The Staley Company, Inc.
Phone: (505) 929-0060

NEW YORK
Ithaca
Bob Dean, Inc.
Phone: (607) 272-2187

NORTH CAROLINA
Cary
Montgomery Marketing
Phone: (919) 467-6319

OHIO
Centerville
Norm Case Associates
Phone: (513) 433-9966
Fairview Park
Norm Case Associates
Phone: (216) 333-4120

OREGON
Portland
Western Technical Sales
Phone: (503) 297-1711

TEXAS
Austin
Cunningham Company
Phone: (512) 459-8947
Dallas
Cunningham Company
Phone: (214) 233-4303
Houston
Cunningham Company
Phone: (713) 461-4197

UTAH
West Bountiful
Barnhill Five, Inc.
Phone: (801) 292-8991

WASHINGTON
Bellevue
Western Technical Sales
Phone: (206) 641-3900

WISCONSIN
Greenfield
L-Tec, Inc.
Phone: (414) 545-8900

DISTRIBUTORS

ALABAMA
Huntsville
Hamilton/Avnet Electronics
Phone: (205) 533-1170

ARIZONA
Phoenix
Hamilton/Avnet Electronics
Phone: (602) 275-7851
Liberty Electronics
Phone: (602) 257-1272

CALIFORNIA
Costa Mesa
Avnet Electronics
Phone: (714) 754-6051
Schweber Electronics
Phone: (714) 556-3880-Culver City
Culver City
Hamilton Electro Sales
Phone: (213) 558-2183
El Segundo
Liberty Electronics
Phone: (213) 322-8100
Mountain View
Elmar Electronics
Phone: (415) 961-3611
Hamilton/Avnet Electronics
Phone: (415) 961-7000
San Diego
Hamilton/Avnet Electronics
Phone: (714) 279-2421
Liberty Electronics
Phone: (714) 565-9171
Sunnyvale
Intermark Electronics
Phone: (408) 738-1111

CANADA
Downsview, Ontario
Cesco Electronics
Phone: (416) 661-0220
Zentronics
Phone: (416) 635-2822
Mississauga, Ontario
Hamilton/Avnet Electronics
Phone: (416) 677-7432
Montreal, Quebec
Cesco Electronics
Phone: (514) 735-5511
Zentronics Ltd.
Phone: (514) 735-5361
Ottawa, Ontario
Cesco Electronics
Phone: (613) 729-5118
Hamilton/Avnet Electronics
Phone: (613) 226-1700
Zentronics Ltd.
Phone: (613) 238-6411
Vancouver, B.C.
Rowtek Electronics Co., Ltd.
Phone: (604) 736-1141
Ville St. Laurent, Quebec
Hamilton/Avnet Electronics
Phone: (514) 331-6443

COLORADO
Commerce City
Elmar Electronics
Phone: (303) 287-9611
Denver
Hamilton/Avnet Electronics
Phone: (303) 534-1212

CONNECTICUT
Danbury
Schweber Electronics
Phone: (203) 792-3500
Georgetown
Hamilton/Avnet Electronics
Phone: (203) 762-0361
Hamden
Arrow Electronics
Phone: (203) 248-3801

FLORIDA
Ft. Lauderdale
Arrow Electronics
Phone: (305) 776-7790
Hamilton/Avnet Electronics
Phone: (305) 971-2800
Hollywood
Schweber Electronics
Phone: (305) 927-0511
Orlando
Hammond Electronics
Phone: (305) 241-6601

GEORGIA
Atlanta
Schweber Electronics
Phone: (404) 449-9170
Norcross
Hamilton/Avnet Electronics
Phone: (404) 448-0800

ILLINOIS
Elk Grove
Schweber Electronics
Phone: (312) 593-2740
Elmhurst
Semiconductor Specialists
Phone: (312) 279-1000
Schiller Park
Hamilton/Avnet Electronics
Phone: (312) 671-6094
Skokie
Bell Industries
Phone: (312) 965-7500

INDIANA
Indianapolis
Pioneer Electronics
Phone: (317) 849-7300

KANSAS
Lenexa
Hamilton/Avnet Electronics
Phone: (913) 888-8900

MARYLAND
Baltimore
Arrow Electronics
Phone: (301) 247-5200
Gaithersburg
Pioneer Washington Electronics
Phone: (301) 948-0710
Hanover
Hamilton/Avnet Electronics
Phone: (301) 796-5000
Rockville
Schweber Electronics
Phone: (301) 881-3300

MASSACHUSETTS
Waltham
Schweber Electronics
Phone: (617) 890-8484
Woburn
Arrow Electronics
Phone: (617) 933-8130
Hamilton/Avnet Electronics
Phone: (617) 933-8000

MICHIGAN
Livonia
Hamilton/Avnet Electronics
Phone: (313) 522-4700
Pioneer Electronics
Phone: (313) 525-1800

MINNESOTA
Eden Prairie
Schweber Electronics
Phone: (612) 941-5280
Edina
Hamilton/Avnet Electronics
Phone: (612) 941-3801
Minneapolis
Semiconductor Specialists
Phone: (612) 854-8841

MISSOURI
Hazelwood
Hamilton/Avnet Electronics
Phone: (314) 731-1144

NEW MEXICO
Albuquerque
Hamilton/Avnet Electronics
Phone: (505) 765-1500

NEW YORK
Buffalo
Summit Distributors
Phone: (716) 884-3450
East Syracuse
Hamilton/Avnet Electronics
Phone: (315) 437-2642
Farmingdale, L.I.
Arrow Electronics
Phone: (516) 694-6800
Rochester
Hamilton/Avnet Electronics
Phone: (716) 442-7820
Schweber Electronics
Phone: (716) 461-4000
Westbury, L.I.
Hamilton/Avnet Electronics
Phone: (516) 333-5800
Schweber Electronics
Phone: (516) 334-7474

NORTHERN NEW JERSEY
Cedar Grove
Hamilton/Avnet Electronics
Phone: (210) 239-0800
Saddlebrook
Arrow Electronics
Phone: (201) 797-5800

SOUTHERN NEW JERSEY AND PENNSYLVANIA
Cherry Hill, N.J.
Milgray-Delaware Valley
Phone: (609) 424-1300
Moorestown, N.J.
Arrow/Angus Electronics
Phone: (609) 235 1900
Mt. Laurel, N.J.
Hamilton/Avnet Electronics
Phone: (609) 234-2133
Pittsburgh, PA
Pioneer/Pittsburgh
Phone: (412) 782-2300

CENTRAL NEW JERSEY AND PENNSYLVANIA
Somerset, N.J.
Schweber Electronics
Phone: (201) 469-6008
Horsham, PA
Schweber Electronics
Phone: (215) 441-0600

NORTH CAROLINA
Greensboro
Hammond Electronics
Phone: (919) 275-6393
Pioneer Electronics
Phone: (919) 273-4441

OHIO
Beechwood
Schweber Electronics
Phone: (216) 464-2970
Cleveland
Hamilton/Avnet Electronics
Phone: (216) 461-1400
Pioneer Standard Electronics
Phone: (216) 587-3600
Dayton
Hamilton/Avnet Electronics
Phone: (513) 439-0610
Pioneer Standard Electronics
Phone: (513) 236-9900

OKLAHOMA
Tulsa
Component Specialties
Phone: (918) 584-7515

TEXAS
Dallas
Component Specialists
Phone: (214) 357-5611
Hamilton/Avnet Electronics
Phone: (214) 661-8204
Quality Components
Phone: (214) 387-4949
Schweber Electronics
Phone: (214) 661-5010
Houston
Component Specialists
Phone: (713) 771-7237
Hamilton/Avnet Electronics
Phone: (713) 780-1771
Quality Components
Phone: (713) 772-7100
Schweber Electronics
Phone: (713) 784-3600

UTAH
Salt Lake City
Alta Electronics
Phone: (801) 486-4134
Hamilton/Avnet Electronics
Phone: (801) 972-2800

WASHINGTON
Bellevue
Hamilton/Avnet Electronics
Phone: (206) 746-8750
Liberty Electronics
Phone: (206) 453-8300

WISCONSIN
New Berlin
Hamilton/Avnet Electronics
Phone: (414) 784-4510

FOR SIGNETICS PRODUCTS WORLDWIDE

ARGENTINA
Fapesa I.y.C.
Buenos-Aires
Phone: 652-7438/7478

AUSTRIA
Osterreichische Philips
Wien
Phone: 93 26 11

AUSTRALIA
Philips Industries-ELCOMA
Lane-Cove N.S.W.
Phone: (02) 427-0888

BELGIUM
M.B.L.E.
Brussels
Phone: 523 00 00

BRAZIL
Ibrape, S.A.
Sao Paulo
Phone: 284-4511

CANADA
Philips Electron Devices
Toronto
Phone: 425-5161

CHILE
Philips Chilena S.A.
Santiago
Phone: 39-4001

COLOMBIA
Sadape S.A.
Bogota
Phone: 600600

DENMARK
Miniwatt A/S
Kobenhavn
Phone: (01) 69 16 22

FINLAND
Oy Philips Ab
Helsinki
Phone: 1 72 71

FRANCE
R.T.C.
Paris
Phone: 355-44-99

W. GERMANY
Nordlingen
Signetics GmbH

HONG KONG
Philips Hong Kong, Ltd.
Hong Kong
Phone: 12-245121

INDIA
Semiconductors, Ltd.
(REPRESENTATIVE ONLY)
Bombay
Phone: 293-667

INDONESIA
P. T. Philips-Ralin Electronics
Jakarta
Phone: 581058

IRAN
Berkeh Company, Ltd.
Tehran
Phone: 831564

ISRAEL
Rapac Electronics, Ltd.
Tel Aviv
Phone: 477115-6-7

ITALY
Philips S.p.A.
Milano
Phone: 2-6994

JAPAN
Signetics Japan, Ltd.
Tokyo
Phone: (03) 230-1521

KOREA
Philips Elect Korea Ltd.
Seoul
Phone: 44-4202

MEXICO
Electronica S.A. de C.V.
Mexico D.F.
Phone: 533-1180

NETHERLANDS
Philips Nederland B.V.
Eindhoven
Phone: (040) 79 33 33

NEW ZEALAND
Philips Electrical Ind. ELCOMA
Auckland

NORWAY
Electronica A.S.
Oslo
Phone: (02) 15 05 90

PAKISTAN
Elmac Ltd
Karachi
Phone: 515-122

PERU
Lima
Cadesa
Phone: 628599

PHILIPPINES
Philips Industrial Dev., Inc.
Makata-Rizal
Phone: 868951-9

SINGAPORE/MALAYSIA
Philips Singapore Pte., Ltd.
Singapore
Phone: 538811

SOUTH AFRICA
E.D.A.C. (PTY), Ltd.
Johannesburg
Phone: 24-6701-3

SPAIN
Copresa S.A.
Barcelona
Phone: 329 63 12

SWEDEN
Elcoma A.B.
Stockholm
Phone: 08/67 97 80

SWITZERLAND
Philips A.G.
Zurich
Phone: 01/44 22 11

TAIWAN
Philips Taiwan, Ltd.
Taipei
Phone: (02) 551-3101-5

THAILAND
Saeng Thong Radio, Ltd.
Bangkok
Phone: 527195, 519763

UNITED KINGDOM
Mullard, Ltd.
London
Phone: 01-580 6633

UNITED STATES
Signetics International Corp.
Sunnyvale, California
Phone: (408) 739-7700

URUGUAY
Luzilector S.A.
Montevideo
Phone: 9143 21

VENEZUELA
Industrias Venezolanas Philips S.A.
Caracas
Phone: 360-511

signetics

a subsidiary of **U.S. Philips Corporation**

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

Printed in USA Oct. 1977