SIGNETICS
8X300
REFERENCE
MANUAL

$3.50
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>3</td>
</tr>
<tr>
<td>Product Description</td>
<td>7</td>
</tr>
<tr>
<td>One-chip Bipolar Microcontroller Approaches Bit-Slice Performance</td>
<td>9</td>
</tr>
<tr>
<td>8X300 Microcontroller Data Sheet</td>
<td>15</td>
</tr>
<tr>
<td>Support Products</td>
<td>31</td>
</tr>
<tr>
<td>Designer's Evaluation Kit for Fixed Instruction Bipolar Microprocessor</td>
<td>33</td>
</tr>
<tr>
<td>8X300 Programming Course</td>
<td>35</td>
</tr>
<tr>
<td>MCCAP 8X300AS100SS</td>
<td>38</td>
</tr>
<tr>
<td>SMS Microcontroller Simulator SMS300</td>
<td>39</td>
</tr>
<tr>
<td>SMS Microcontroller MS3300</td>
<td>41</td>
</tr>
<tr>
<td>Application Memos</td>
<td>43</td>
</tr>
<tr>
<td>8X300 Input/Output Design AH3</td>
<td>45</td>
</tr>
<tr>
<td>8X300 Applications AH4</td>
<td>47</td>
</tr>
<tr>
<td>Signetics 8X300 CRT Controller</td>
<td>62</td>
</tr>
<tr>
<td>8X300 Cross Assembly Program (MCCAP) SPI</td>
<td>83</td>
</tr>
<tr>
<td>8X300 Program Library B3</td>
<td>89</td>
</tr>
<tr>
<td>Introduction</td>
<td>89</td>
</tr>
<tr>
<td>TAD16 Procedure</td>
<td>90</td>
</tr>
<tr>
<td>MUL8X8 Procedure</td>
<td>91</td>
</tr>
<tr>
<td>M16X16 Procedure</td>
<td>92</td>
</tr>
<tr>
<td>DIV16X8 Procedure</td>
<td>94</td>
</tr>
<tr>
<td>D24X12 Procedure</td>
<td>96</td>
</tr>
<tr>
<td>DTOB Procedure</td>
<td>98</td>
</tr>
<tr>
<td>BTOD Procedure</td>
<td>100</td>
</tr>
<tr>
<td>SORT Procedure</td>
<td>102</td>
</tr>
<tr>
<td>FPADD Procedure</td>
<td>103</td>
</tr>
<tr>
<td>DSHIFT Procedure</td>
<td>105</td>
</tr>
<tr>
<td>TAD16F Procedure</td>
<td>107</td>
</tr>
<tr>
<td>Data Sheets</td>
<td>109</td>
</tr>
<tr>
<td>8T31</td>
<td>111</td>
</tr>
<tr>
<td>8T32/8T33/8T35/8T36</td>
<td>115</td>
</tr>
<tr>
<td>8T39</td>
<td>122</td>
</tr>
<tr>
<td>8T58</td>
<td>126</td>
</tr>
<tr>
<td>Bipolar Memory Selection Guide</td>
<td>129</td>
</tr>
<tr>
<td>82S114/82S115</td>
<td>131</td>
</tr>
<tr>
<td>82S16/82S116—82S17/82S117</td>
<td>136</td>
</tr>
<tr>
<td>Appendix</td>
<td>141</td>
</tr>
<tr>
<td>8X300 Designer's Evaluation Kit Instruction Manual 8X300KT100SK</td>
<td>143</td>
</tr>
<tr>
<td>Sales Offices</td>
<td>151</td>
</tr>
</tbody>
</table>
PRODUCT DESCRIPTION
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
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 (AR0 through AR4), or from the output of the arithmetic/logic unit (lines AR5 through AR11). 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—R1 through R6, R11, 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 IV0 through IV7. 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...
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...
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
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 LB and 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 DESIGNATION

<table>
<thead>
<tr>
<th>PIN</th>
<th>SYMBOL</th>
<th>NAME AND FUNCTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-115</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-15</td>
<td>AO-A12</td>
<td>Instruction address lines. A high level equals “1.” These outputs directly address up to 8192 words of program storage. A12 is least significant bit.</td>
<td>Active high</td>
</tr>
<tr>
<td>13-28</td>
<td>I0-I15</td>
<td>Instruction lines. A high level equals “1.” Receives instructions from Program Storage. I15 is least significant bit.</td>
<td>Active high</td>
</tr>
<tr>
<td>33-36, 36-41</td>
<td>IVBO-IVB7</td>
<td>Interface Vector (IV) Bus. A low level equals “1.” Bidirectional tri-state lines to communicate with I/O devices. IVB7 is least significant bit.</td>
<td>Three-state</td>
</tr>
<tr>
<td>42</td>
<td>MCLK</td>
<td>Master Clock. Output to clock I/O devices, and/or provide synchronization for external logic</td>
<td>Active high</td>
</tr>
<tr>
<td>30</td>
<td>WC</td>
<td>Write Command. High level output indicates data is being output on the IV Bus.</td>
<td>Active high</td>
</tr>
<tr>
<td>29</td>
<td>SC</td>
<td>Select Command. High level output indicates that an address is being output on the IV Bus.</td>
<td>Active high</td>
</tr>
<tr>
<td>31</td>
<td>LB</td>
<td>Left Bank. Low level output to enable one of two sets of I/O devices (LB is the complement of RB).</td>
<td>Active low</td>
</tr>
<tr>
<td>32</td>
<td>RB</td>
<td>Right Bank. Low level output to enable one of two sets of I/O devices (RB is the complement of LB).</td>
<td>Active low</td>
</tr>
<tr>
<td>44</td>
<td>HALT</td>
<td>Low level is input to stop the Microcontroller.</td>
<td>Active low</td>
</tr>
<tr>
<td>43</td>
<td>RESET</td>
<td>Low level is input to initialize the Microcontroller.</td>
<td>Active low</td>
</tr>
<tr>
<td>10-11</td>
<td>X1, X2</td>
<td>Inputs for an external frequency determining crystal. May also be interfaced to logic or test equipment.</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>VR</td>
<td>Reference voltage to pass transistor.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VCR</td>
<td>Regulated output voltage from pass transistor.</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>VCC</td>
<td>5V power connection.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>GND</td>
<td>Ground.</td>
<td></td>
</tr>
</tbody>
</table>
**Microcontroller Architecture**

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

---

**Figure 1**

**Table 1 INTERNAL REGISTERS**

---

**Notes:**
- The least-significant bit of the OVF register is used to reflect overflow status resulting from the most recent ADD operation (see Instruction Set Summary).
- Normally contains the address of the current instruction and is incremented to obtain the next instruction address.
- A 13-bit register containing the address of the current instruction.
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 IO-115 during the first quarter cycle of the next instruction cycle (see Table 2).

Instruction Set Summary
The 16-bit instruction word input on IO-115 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:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Class</th>
<th>Operands Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 15 operand(s) specification bits. A complete description of instruction formats and some instruction examples are provided in the Microprocessor Application 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.

IIVL and IIVR 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 IIVL is specified as a destination or the D field = 20-27b, then LB = 'low', RB = 'high' are generated; when IIVR is specified as a destination or the D field = 30-37b, then RB = low, LB = 'high' are generated.

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

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 represented 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.

### Table 3 8X300 INSTRUCTION SUMMARY

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>FORMAT</th>
<th>RESULT</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>I,II</td>
<td>Content of data field specified by {S, R/L} replaces data in field specified by {D, R/L}</td>
<td>If S and D both are registers, then R/L specifies a right rotate of the register specified by S.</td>
</tr>
<tr>
<td>ADD</td>
<td>I,II</td>
<td>Sum of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L}</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>I,II</td>
<td>Logical AND of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L}</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>I,II</td>
<td>Logical exclusive OR of AUX and data specified by {S, R/L} replaces data in field specified by {D, R/L}</td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>III,IV</td>
<td>The literal value I replaces the data in the field specified by {S.L}</td>
<td>If S is an I/O address then I is limited to range 00-37. Otherwise I is limited to range 000-377.</td>
</tr>
<tr>
<td>NZT</td>
<td>III,IV</td>
<td>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.</td>
<td>If S is an I/O address then I is limited to range 00-37. Otherwise I is limited to range 000-377.</td>
</tr>
<tr>
<td>XEC</td>
<td>III,IV</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>JMP</td>
<td>V</td>
<td>The address value A replaces contents of the Program Counter.</td>
<td>A limited to the range 0-17777s.</td>
</tr>
</tbody>
</table>

*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.

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>OCTAL VALUE</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>MOVE S,R/L,D</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>ADD S,R/L,D</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>AND S,R/L,D</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>XOR S,R/L,D</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>XEC I,R/L,S or I,S</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>NZT I,R/L,S or I,S</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>XMIT I,R/L,D or I,D</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>JMP</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>OCTAL VALUE</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUX-Auxiliary Register</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
<td>R5</td>
<td>R6</td>
<td>IVL Left Bank I/O address register</td>
</tr>
<tr>
<td></td>
<td>OVF-Overflow register-used only as a source</td>
<td>R11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unassigned</td>
<td>Unassigned</td>
<td>Unassigned</td>
<td>Unassigned</td>
<td>Unassigned</td>
<td>Unassigned</td>
<td>IVL Right Bank I/O address register-used only as a destination</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>R/L FIELD OCTAL VALUE</th>
<th>SPECIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Field Length = 8 Bits</td>
</tr>
<tr>
<td>1</td>
<td>Field Length = 1 Bit</td>
</tr>
<tr>
<td>2</td>
<td>Field Length = 2 Bits</td>
</tr>
<tr>
<td>3</td>
<td>Field Length = 3 Bits</td>
</tr>
<tr>
<td>4</td>
<td>Field Length = 4 Bits</td>
</tr>
<tr>
<td>5</td>
<td>Field Length = 5 Bits</td>
</tr>
<tr>
<td>6</td>
<td>Field Length = 6 Bits</td>
</tr>
<tr>
<td>7</td>
<td>Field Length = 7 Bits</td>
</tr>
</tbody>
</table>

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 0-25510.

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.

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

**Format:** Type I, Type II  
**Operation:** $(S) \land (AUX) \rightarrow D$  
**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.

---

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

**Format:** Type I, Type II  
**Operation:** $(S) \oplus (AUX) \rightarrow D$  
**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.
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.

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.
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.

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 0-177778 (0-8191).
Example
Jump to location ALPHA (0000101110001). See Figure 11.
SYSTEM DESIGN USING THE 8X300 MICROCONTROLLER

Designing hardware around the 8X300 Interpreter reduces to selecting a program storage device (ROM, PROM, etc.), selecting I/O devices (I/O 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**

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

**Reset:**
A low level at the RESET input sets the program counter and address register to zero. While RESET is low MCLK is inhibited. If RESET is applied during the last 2 quarter cycles, the MCLK during that cycle may be shortened. RESET should be applied for 1 full instruction cycle time to assure proper operation. When RESET input goes high an MCLK occurs prior to the resumption of normal processing. 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.

---

**Figure 12**

---

**Signetics**
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: $T_{OE} \leq \frac{1}{2}$ cycle - 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 $\leq$ instruction cycle time (Figure 14).
3. Program storage access time + instruction to address $\leq$ instruction cycle time (Figure 15).

The first constraint can be used to determine the minimum cycle time. Using the inequality $35ns + 35ns \leq \frac{1}{2}$ cycle - 55ns implies $\frac{1}{2}$ cycle $\geq 125ns$ 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 $\leq$ 250ns implies program storage access time $\leq$ 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 $\leq$ 250ns confirms that program storage access time 65ns is satisfactory.
System Clock
The Microcontroller 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Fundamental mode, series resonant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impedance at Fundamental</td>
<td>35 ohms maximum</td>
</tr>
<tr>
<td>Impedance at harmonics and spurs:</td>
<td>50 ohms minimum</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Cx,pF</th>
<th>CYCLE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>300ns</td>
</tr>
<tr>
<td>200</td>
<td>500ns</td>
</tr>
<tr>
<td>500</td>
<td>1.1μs</td>
</tr>
<tr>
<td>1000</td>
<td>2.0μs</td>
</tr>
</tbody>
</table>

Table 8 CLOCK CAPACITOR VALUES

CLOCKING WITH A PULSE GENERATOR

CLOCKING WITH TTL

AC ELECTRICAL CHARACTERISTICS $V_{CC} = 5V \pm 5\%$ and $0^\circ C \leq T_A < 70^\circ C$

<table>
<thead>
<tr>
<th>DELAY DESCRIPTION</th>
<th>PROPAGATION DELAY TIME</th>
<th>CYCLE TIME LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1 falling edge to MCLK (driven from external pulse generator)</td>
<td>75ns</td>
<td>25ns</td>
</tr>
<tr>
<td>MCLK to SC/WC falling edge (input phase)</td>
<td>25ns</td>
<td>25ns</td>
</tr>
<tr>
<td>MCLK to SC/WC rising edge (output phase)</td>
<td>35ns</td>
<td>35ns</td>
</tr>
<tr>
<td>MCLK to LB/RB (input phase)</td>
<td>35ns</td>
<td>35ns</td>
</tr>
<tr>
<td>Instruction to LB/RB output (input phase)</td>
<td>35ns</td>
<td>35ns</td>
</tr>
<tr>
<td>MCLK to LB/RB (output phase)</td>
<td>185ns</td>
<td>185ns</td>
</tr>
<tr>
<td>MCLK to IV data (output phase)</td>
<td>185ns</td>
<td>185ns</td>
</tr>
<tr>
<td>IV data (input phase) to IV data (output phase)</td>
<td>185ns</td>
<td>185ns</td>
</tr>
<tr>
<td>Instruction to Address</td>
<td>115ns</td>
<td>115ns</td>
</tr>
<tr>
<td>MCLK to Address</td>
<td>115ns</td>
<td>115ns</td>
</tr>
<tr>
<td>IV data (input phase) to Address</td>
<td>115ns</td>
<td>115ns</td>
</tr>
<tr>
<td>MCLK to IV data (input phase)</td>
<td>115ns</td>
<td>115ns</td>
</tr>
<tr>
<td>MCLK to Halt falling edge to prevent current cycle</td>
<td>115ns</td>
<td>115ns</td>
</tr>
<tr>
<td>Reset rising edge to first MCLK</td>
<td>115ns</td>
<td>115ns</td>
</tr>
</tbody>
</table>

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

- INSTRUCTION CYCLE
  - INPUT PHASE
  - OUTPUT PHASE
  - INPUT PHASE

- INST & IV BUS
- INPUTS ACCEPTED
- DATA PROCESSING
- ADDRESS & IV BUS CHANGING
- ADDRESS & IV BUS OUTPUT STABLE

MCLK

INSTRUCTION ADDRESS
(M0-M15)

INSTRUCTION DATA
(M0-M15)

SC/WC

LB/RB

IV BUS
(IVBO-IVB7)

Figure 18
### DC ELECTRICAL CHARACTERISTICS

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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$ High level input voltage</td>
<td>X1,X2</td>
<td>Min</td>
<td>Typ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low level input voltage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input clamp voltage</td>
<td>V_{CC} = 4.75V, I_{I} = -10mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{IH}$ High level input current</td>
<td>X1,X2</td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.25V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{IH} = .6V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{CC} = 5.25V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{IH} = 4.5V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;1</td>
</tr>
<tr>
<td>Low level input current</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$ Low level input current</td>
<td>X1,X2</td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.25V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{IL} = .4V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{CC} = 5.25V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{IL} = .4V$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-140</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-230</td>
</tr>
<tr>
<td>Low level output voltage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OL}$ Low level output voltage</td>
<td>A0-A12</td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.75V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I_{OL}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.25mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.75V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I_{OL}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.75V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High level output voltage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{OS}$ Short circuit output current</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.25V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply voltage</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>5</td>
</tr>
<tr>
<td>Supply current</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.25V</td>
<td></td>
</tr>
<tr>
<td>Regulator control</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0V</td>
<td></td>
</tr>
<tr>
<td>Regulator current</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.25V</td>
<td></td>
</tr>
<tr>
<td>Regulator voltage</td>
<td></td>
<td>V_{CC}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.2</td>
</tr>
</tbody>
</table>

**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.
SUPPORT PRODUCTS
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 masked before undergoing arithmetic or logical operations. Then, the data may be shifted and merged into any set of from 1 to 6 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 interconnected 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 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 Select)
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
1 ea- Introductory Manual, assembly instructions, code listings and schematics

AVAILABILITY
Immediate delivery from Signetics Rep. or Distributors.
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**
  - I/O 8T32/33 82S115 (512x8 PROM)
  - I/O 8T35/36 (8-Bit Synchronous Bidirectional I/O Port)
  - I/O 8T31 (8-Bit Bidirectional I/O Port)
  - I/O 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:
   825215 bipolar ROM
   825115 bipolar PROM
   825116 bipolar RAM
   8T32/33 IV Byte
3. 256-byte working storage configuration.
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
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 (i) 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
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.

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
Completion: Approximately 45 minutes

Lesson 7—8X300 Arithmetic/Logic Commands
With prior emphasis on 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 nonmicroprocessor 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
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 operands and translation tables.

Audio Tape: Approximately 35 minutes
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 application.
5. Record keeping for I/O, process, and address control.
7. Effect of poor record keeping applications on CPU program execution.
8. Techniques for effective record keeping.
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/absolute 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. Clocking 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 variable 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 two 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. RETURN ADDRESS STACK definition and architecture
2. Software generation of RAS: Usage of working storage
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

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
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 rewritable 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-defined 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 instructions 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.
- 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 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.
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/keypad 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 1 and 1 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).

<table>
<thead>
<tr>
<th>MODE</th>
<th>FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>Change Address</td>
</tr>
<tr>
<td></td>
<td>Store Instruction</td>
</tr>
<tr>
<td>Tape</td>
<td>Load</td>
</tr>
<tr>
<td></td>
<td>Verify</td>
</tr>
<tr>
<td></td>
<td>Begin Punch Address</td>
</tr>
<tr>
<td></td>
<td>End Punch Address</td>
</tr>
<tr>
<td></td>
<td>Punch</td>
</tr>
<tr>
<td></td>
<td>Program Identification</td>
</tr>
<tr>
<td>Register</td>
<td>Change Address</td>
</tr>
<tr>
<td></td>
<td>Store Octal Data</td>
</tr>
<tr>
<td></td>
<td>Store Binary Data</td>
</tr>
<tr>
<td></td>
<td>Complement Overflow</td>
</tr>
<tr>
<td>Interface Vector</td>
<td>Display Currently Active</td>
</tr>
<tr>
<td></td>
<td>Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Set Normal (Sync) Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Set Stop Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Replace Instruction at Breakpoint</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Breakpoint</td>
<td>Display Currently Active</td>
</tr>
<tr>
<td></td>
<td>Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Set Normal (Sync) Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Set Stop Breakpoint</td>
</tr>
<tr>
<td></td>
<td>Replace Instruction at Breakpoint</td>
</tr>
<tr>
<td>Execute</td>
<td>Halt</td>
</tr>
<tr>
<td></td>
<td>Run Program</td>
</tr>
<tr>
<td></td>
<td>Insert Instruction</td>
</tr>
<tr>
<td></td>
<td>Single Step</td>
</tr>
</tbody>
</table>

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:

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

Data Input/Output

Physical Characteristics

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

<table>
<thead>
<tr>
<th>Controls</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN/STEP</td>
<td>Sets operation mode, continuous execution (RUN) or single step (STEP).</td>
</tr>
<tr>
<td>INSERT INST</td>
<td>Unconditionally causes execution of the instruction in the Instruction Switch Register.</td>
</tr>
<tr>
<td>RESET/HALT</td>
<td>Three position switch used to unconditionally HALT program execution or RESET Interpreter's Program Counter to zero.</td>
</tr>
<tr>
<td>PGM ADR BKPT</td>
<td>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.</td>
</tr>
<tr>
<td>IV BKPT</td>
<td>Selects appropriate function to be performed when IV Breakpoint is reached: HALT—halt on breakpoint, OFF—program execution unchanged by breakpoint.</td>
</tr>
<tr>
<td>BKPT UPON</td>
<td>Selects IV Breakpoint on one of three conditions: specified IV address, specified write data at IV address, specified read data at the IV address.</td>
</tr>
<tr>
<td>SYNC SOURCE</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

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

**Dimensions**

- Power 115Vac ± 10% at 0.75 amp max. |
- 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 connection 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 ADDRESS, IVL
XM1T
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 0-255.

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.

Table 1 FUNCTIONS OF THE BIC AND BOC LINES

<table>
<thead>
<tr>
<th>CONTROL LINES</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOC</td>
<td>BIC</td>
</tr>
<tr>
<td>(low true)</td>
<td>(low true)</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>X</td>
<td>L</td>
</tr>
</tbody>
</table>

8 I/O lines in high impedance state—disable
8 I/O lines in output mode—8 bit storage latch data available in the output lines.
8 I/O lines in input mode—data can be read by Interpreter.

Table 2 IV BYTE TERMINAL ELECTRICAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>VIH</td>
<td>2</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>VIL</td>
<td>-1</td>
<td>0.8</td>
<td>V</td>
</tr>
<tr>
<td>VIC</td>
<td>2.4</td>
<td>0.5</td>
<td>V</td>
</tr>
<tr>
<td>VOH</td>
<td>-20</td>
<td>-200</td>
<td>mA</td>
</tr>
<tr>
<td>VOL</td>
<td>10</td>
<td>1000</td>
<td>uA</td>
</tr>
<tr>
<td>IOS</td>
<td>12</td>
<td>12</td>
<td>pF</td>
</tr>
</tbody>
</table>

NOTE
1. Input current is always present regardless of the state of BIC and BOC.
8X300 TYPICAL SYSTEM CONFIGURATION WITH WORKING STORAGE

Figure 4
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, inputting 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).
## 8X300 Applications

### Table 1: Interface Analysis

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Data Rate</th>
<th>Signal Duration</th>
<th>Electrical Characteristics</th>
<th># IV Bits</th>
<th>Interface Required</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step In</td>
<td>20 ms</td>
<td>0.1-10 ms</td>
<td>TTL with pullup</td>
<td>1</td>
<td></td>
<td>Step head 1 track in</td>
</tr>
<tr>
<td>Step Out</td>
<td>20 ms</td>
<td>0.1-10 ms</td>
<td>TTL with pullup</td>
<td>1</td>
<td></td>
<td>Step head 1 track out</td>
</tr>
<tr>
<td>Load Head</td>
<td>Level</td>
<td></td>
<td>TTL with pullup</td>
<td>1</td>
<td></td>
<td>Load head</td>
</tr>
<tr>
<td>Unsafe Reset</td>
<td>Level</td>
<td></td>
<td>TTL with pullup</td>
<td>1</td>
<td></td>
<td>Clears unsafe condition</td>
</tr>
<tr>
<td>WR ENB</td>
<td>Level</td>
<td></td>
<td>TTL with pullup</td>
<td>1</td>
<td></td>
<td>Enables write operation</td>
</tr>
<tr>
<td>WR Data</td>
<td>2 μs</td>
<td>25 μs</td>
<td>50 mA current</td>
<td>1</td>
<td>2R, T</td>
<td>Data/clock to disc</td>
</tr>
<tr>
<td>Sector</td>
<td>5 ms</td>
<td>1 ms</td>
<td>OC output</td>
<td>1</td>
<td>R</td>
<td>Sector indicated</td>
</tr>
<tr>
<td>Index</td>
<td>160 ms</td>
<td>1 ms</td>
<td>OC output</td>
<td>1</td>
<td>R</td>
<td>Begin of track indicator</td>
</tr>
<tr>
<td>TRK00</td>
<td>Level</td>
<td></td>
<td>OC output</td>
<td>1</td>
<td>R</td>
<td>Head on track 00</td>
</tr>
<tr>
<td>Unsafe</td>
<td>Level</td>
<td></td>
<td>OC output</td>
<td>1</td>
<td>R</td>
<td>Unsafe condition indicator</td>
</tr>
<tr>
<td>WR Protect</td>
<td>Level</td>
<td></td>
<td>OC output</td>
<td>1</td>
<td>R, 2FF</td>
<td>Write protected disc</td>
</tr>
<tr>
<td>Data</td>
<td>4 μs</td>
<td>25 μs</td>
<td>OC output</td>
<td>1</td>
<td>R, FF</td>
<td>Data from disc</td>
</tr>
<tr>
<td>Clock</td>
<td>4 μs</td>
<td>25 μs</td>
<td>OC output</td>
<td>1</td>
<td></td>
<td>Clock from disc</td>
</tr>
</tbody>
</table>

R = Resistor  
T = Transistor  
FF = Flip-Flop

### Table 2: 8X300 Configuration

<table>
<thead>
<tr>
<th>ROM/PROM for Program Storage</th>
<th>Working Storage for Data Buffers</th>
<th>IV Bytes for Input/Output Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Driver</td>
<td>76 words</td>
<td>256 Bytes</td>
</tr>
<tr>
<td>Output Driver</td>
<td>74 words</td>
<td></td>
</tr>
<tr>
<td>Head Step Driver</td>
<td>30 words</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>180 words</td>
<td></td>
</tr>
</tbody>
</table>

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
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).

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DATA RATE</th>
<th>SIGNAL DURATION</th>
<th>ELECTRICAL CHARACTERISTICS</th>
<th># IV BYTES AS REQUIRED</th>
<th>INTERFACE REQUIRED</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTY OUT</td>
<td>9.09ms</td>
<td>9.09ms</td>
<td>20mA current</td>
<td>1</td>
<td>3R,T</td>
<td>Data to TTY printer</td>
</tr>
<tr>
<td>TTY IN</td>
<td>9.09ms</td>
<td>9.09ms</td>
<td>20mA current</td>
<td>1</td>
<td>4R,T,C</td>
<td>Data from TTY keyboard</td>
</tr>
</tbody>
</table>

R = Resistor
T = Transistor
C = Capacitor

Figure 3

Table 3 INTERFACE ANALYSIS
FUNCTIONAL ANALYSIS

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

Figure 4

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

Table 4 8X300 CONFIGURATION
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).
Table 5  INTERFACE ANALYSIS

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

Figure 6

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

Table 6 8X300 CONFIGURATION

<table>
<thead>
<tr>
<th>ROM/PROM FOR PROGRAM STORAGE</th>
<th>WORKING STORAGE FOR DATA BUFFERS</th>
<th>IV BYTES FOR INPUT/OUTPUT INTERFACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplexer driver</td>
<td>156 words</td>
<td>13 IV bits for output per UART</td>
</tr>
<tr>
<td>Character processing</td>
<td>100 words</td>
<td>15 IV bits for input per UART</td>
</tr>
<tr>
<td>Total</td>
<td>256 words</td>
<td>Total: 4 IV bytes per UART</td>
</tr>
</tbody>
</table>

Through put: (a) Peak data rate: 2880 characters/sec. (b) Peak processor utilization 30% including modem servicing.
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).
### 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 | 200ns (min) | TTL | 3 | - | Select row of digit
DIGIT SEL | level | | TTL | 32 | - | Select digit for display

Table 7 INTERFACE ANALYSIS

### FUNCTIONAL ANALYSIS

Figure 8

Table 8 8X300 CONFIGURATION

<table>
<thead>
<tr>
<th>ROM/PROM FOR PROGRAM STORAGE</th>
<th>WORKING STORAGE FOR DATA BUFFERS</th>
<th>IV BYTES FOR INPUT/OUTPUT INTERFACE</th>
</tr>
</thead>
</table>
| 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 |
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).

Figure 9
### Table 9 INTERFACE ANALYSIS

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DATA RATE</th>
<th>SIGNAL DURATION</th>
<th>ELECTRICAL CHARACTERISTICS</th>
<th># IV BITS</th>
<th>INTERFACE REQUIRED</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-07</td>
<td>level</td>
<td></td>
<td>open collector, TTL</td>
<td>8</td>
<td>8D</td>
<td>Data/address from computer</td>
</tr>
<tr>
<td>0000-07</td>
<td>level</td>
<td></td>
<td>open collector</td>
<td>8</td>
<td>8R</td>
<td>Data to computer</td>
</tr>
<tr>
<td>COXX</td>
<td>4μs</td>
<td>1.25μs</td>
<td>open collector</td>
<td>1</td>
<td>D</td>
<td>Control output timing</td>
</tr>
<tr>
<td>DIXX</td>
<td>4μs</td>
<td>1.25μs</td>
<td>open collector</td>
<td>1</td>
<td>D</td>
<td>Data input timing</td>
</tr>
<tr>
<td>DOXX</td>
<td>4μs</td>
<td>0.75-1.25μs</td>
<td>open collector</td>
<td>1</td>
<td>D</td>
<td>Data output timing</td>
</tr>
</tbody>
</table>

D = Open collector driver  
R = Resistors

### Table 10 8X300 CONFIGURATION

<table>
<thead>
<tr>
<th>ROM/PROM FOR PROGRAM STORAGE</th>
<th>WORKING STORAGE FOR DATA BUFFERS</th>
<th>IV BYTES FOR INPUT/OUTPUT INTERFACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Driver ........................</td>
<td>Depends on peripheral</td>
<td>11 IV bits for output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 IV bits for input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total: 3 IV bytes per peripheral</td>
</tr>
</tbody>
</table>

Figure 10
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).
### Table 11 INTERFACE ANALYSIS

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

### GET INSTRUCTION SEQUENCE

1. **STORE MEMORY ADDRESS**
   - XMIT ADRHI, IVL
   - MOVE “ADDRESS HI”, ADRHI
   - XMIT ADRLO, IVL
   - MOVE “ADDRESS LOW”, ADRLO

2. **SET READ CONTROL BIT**
   - XMIT CNTRL, IVL
   - XMIT 5, CNTRL
   - XMIT 7, CNTRL
   - XMIT DATA, IVL

3. **DELAY 250ns**

4. **READ DATA BYTE**
   - DATA AVAILABLE FOR PROCESSING
   - •
   - •
   - •

**Figure 12**
PUT INSTRUCTION SEQUENCE

- STORE MEMORY ADDRESS
- STORE DATA IN IV BYTE
- SET WRITE CONTROL BIT
- DELAY 250ns
- XMIT ADRHI, IVL
  MOVE "ADDRESS HI", ADRHI
  XMIT ADRLO, IVL
  MOVE "ADDRESS LOW", ADRLO
- ADDRESS THE MEMORY
- STORE DATA, IVL
  MOVE "DATA", DATA
  XMIT CNTRL, IVL
  XMIT 2, CNTRL
  XMIT 3, CNTRL
  XMIT 7, CNTRL
- WRITE DATA
- CONTINUE PROCESSING

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

<table>
<thead>
<tr>
<th>ROM/PROM FOR PROGRAM STORAGE</th>
<th>WORKING STORAGE</th>
<th>IV BYTES FOR INPUT/OUTPUT INTERFACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET sequence ..................... 4 words</td>
<td>None</td>
<td>18 IV bits for output</td>
</tr>
<tr>
<td>PUT sequence ..................... 6 words</td>
<td>8 IV bits for input and output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total: 4 IV bytes</td>
</tr>
</tbody>
</table>

Table 12 8X300 CONFIGURATION
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.

NOTE

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

Figure 14
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.
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 PCI2651 (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 OIL 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.

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 OIL 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.
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

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.
TIMING, CONTROL AND CHARACTER DOT GENERATION
(82S115 AS CHARACTER ROM)

Figure 4
MICROPROCESSOR AND PROGRAM STORE

Figure 6
The memory has a size of 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. The necessary latches for both data and control in a 8X300 system. Because data must be routed from and to the PCI, an BT32 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 BT15 and BT16 level converters are used.

The PCI needs a clock of 5.0668MHz. 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 Figures 6 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 45.7μs = 0.571μs = 571ns.

The 11-bit address counting also has to be done within 571ns. Double byte counting in IV bytes would cost at least 8 8X300 cycles per address step, or 6250ns = 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 position 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 ADRHG,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 1.75 μs = 163ns, where the tACC (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). The line synchronization (LSYNCHN) is active “0” for 8 character positions (8 X 571ns = 4.57μs). The line end blanking starts 8 characters after the line synchronization and is active during 32 character positions (32 X 571ns = 18.5μs). The line end blanking inhibits the dot display and the character counting.

**FRAME SYNCHRONIZATION**

Frame synchronization is controlled by 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 (LCTRL2) 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.

**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 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 X300 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 8, the frame row counter and the refresh row counter (FROW and REFR) registers in the X300 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 serviced. The 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. CMDSTRO 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
KEYBDDO is the keyboard 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 and released. In this way, only 1 break pulse can be transmitted per key strobe. The keystrobe 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.

Figure 9

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 X300 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 8, the frame row counter and the refresh row counter (FROW and REFR) registers in the X300 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. CMDSTRO 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
KEYBDDO is the keyboard 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 keystrobe 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.

Figure 9

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 X300 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 8, the frame row counter and the refresh row counter (FROW and REFR) registers in the X300 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. CMDSTRO 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
KEYBDDO is the keyboard 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 keystrobe 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.

Figure 9

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 X300 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 8, the frame row counter and the refresh row counter (FROW and REFR) registers in the X300 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. CMDSTRO 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
KEYBDDO is the keyboard 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 keystrobe 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.

Figure 9

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 X300 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 8, the frame row counter and the refresh row counter (FROW and REFR) registers in the X300 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. CMDSTRO 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
KEYBDDO is the keyboard 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 keystrobe 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.

Figure 9
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 FOP/HOP 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 CMOR 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.

**PCINI**
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.

**COMMAND ROUTINES**
Command routines are started from the main scan loop with an indexed jump. The command register CDMR 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).

**CHRSRO**
CHRSRO 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.

CURSR\(\), Cursor right routine
CURSDW Cursor down routine
CURSLE Cursor left routine
CURSUP Cursor up routine
CRETO Carriage return or cursor return routine
CUSHO Cursor home routine

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.

CLESCR
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.

CLRIN
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 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 CRLNS command if a complete row is cleared.
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 equivalents, 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

CLRSRC

# TO CROW
4 TO CPOS

FROW LT 25 ?

NO

YES

FROW TO REFR

24 TO REFR

CLOSOF

FROW LT 25 ?

NO

YES

REFR - FROW
TO CHAR

REFR TO CHAR

CHAR LT 8 ?

YES

NO

CHAR = 25

CPOS TO MEMORY

CBLOCK

5 + CHAR - CURSOR X (BIT 11)
TO CPOS

SCR = 1

CLEAR MEMORY POSITION:
LOWER BITS IN SCR
HIGHER BITS IN CPOS

END OF BLOCK ?

NO

YES

END OF ROW ?
(LOWER BITS.EQ.4)

NO

YES

SET CLRLNS INDEX
IN CMDR

LINEND

CLRLNS

CHAR .EQ. CROW ?

YES

NO

CHAR = 25

RESTORE CPOS
FROM MEMORY

25 TO CHAR

COMEND

CHAR = 1

RESET DTR IN PIO

SET CMDR TO CBLOCK

80 TO SCR

LINEND

Figure 17
APPENDIX 1

8X300 CRT Controller Package Count

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

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 into 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 45us. With an ROM access time of 600ns + latch + SR setup 45,000ns = 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 51us).
2. Selection of the 2513 on an access time of 555ns 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.
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

CLK1
CAR1
CLK2
LENDBN
CHRCLK
CHRADR
RAM DATA
LATCH DATA SETUP
LATCH CLK TO OUTPUT
ROM ACCESS

RAM timing
Counter CLK to RAM address 74162 99ns
RAM address to RAM data 21L00-3 400ns
Character latch setup time 8ns
Total 507ns (available 571ns)

ROM timing
Latch CLK to data out 74S174/5175 CASE A CASE B
ROM address to output 2513 17ns 17ns
Shift register data setup 74166 534ns 600ns
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
Figure 20
RAM AND ROM TIMING DIAGRAM

RAM timing
Counter CLK to RAM address 74162 69ns
RAM address to RAM data 21L00-3 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 45.7 μs). 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).

![JMP ALPHA](image)

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 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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 compilation. This includes the original source code and error messages. 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.

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
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:

\[
0.5\mu s \leq \text{Entry Time} \leq 4.0\mu s
0.5\mu s \leq \text{Exit Time} \leq 4.0\mu s
\]

Details of the code required for procedure
CALL and RTN are provided in the Pro-
gramming 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 in-
voked any number of times in the program.
This facility simplifies the coding of pro-
grams, 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 state-
ment 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 defini-
tion.

**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
eamples. Detailed use of the instructions
including rules of syntax and parameter
restrictions are described in the MCCAP

**Assembler Directive Statements**
Assembler Directive statements define
program structure and control the assem-
bler outputs. They do not result in the gen-
eration of 8X300 executable code. There are
twelve Assembler Directive statements:

\[
\begin{align*}
\text{PROG Statement} \\
\text{PROC Statement}
\end{align*}
\]

**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 com-
plete program.

**Examples:** END SUB1
END MAIN

**ORG Statement**
Use
Sets the program counter to the value speci-
fied 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 for-
mat.

**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 to the contents of the AUX register. Thelogical 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 -0001011B,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 WS variable or IV variable, making the result immediately available to the user's system.

JMP Statement

Use
To transfer execution of the program to a 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.

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 \((A + B) \oplus (A - B)\) is logically equivalent to \(A \oplus B\).

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 200p.

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.

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, Memory to Memory
Add a 16-bit value in Working Storage, OPERAND1, to a 16-bit value in Working Storage, OPERAND2, and store the result in R1. R1 contains the high order bytes of the result. The MOV instruction is executed when STROBE = 1. Loop until STROBE = 1.

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 INPB/IT, is used as the bit data interface. Figure 10 illustrates the byte assembly.

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.

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 \(R1 > R2\).
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.

Control Sequence #1

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

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

<table>
<thead>
<tr>
<th>XMIT</th>
<th>STATUS/WL</th>
<th>Choose the IV byte which receives the 3-bit input from user's system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>STATUS/R1</td>
<td>Move the 3 bits of interest from the IV byte to register R1. The 3 bits are right justified.</td>
</tr>
<tr>
<td>XMIT</td>
<td>ALARM/IV</td>
<td>Choose the IV byte through which the response is sent to the user's system.</td>
</tr>
<tr>
<td>XEC</td>
<td>PATTERN(R1);B</td>
<td>Select specific pattern from PATTERN table.</td>
</tr>
<tr>
<td>JMP</td>
<td>*+9</td>
<td></td>
</tr>
</tbody>
</table>

#### PATTERN
- A, ALARM
- B, ALARM
- C, ALARM
- • • • H, ALARM

Transmit proper pattern to output IV byte subfield by executing just one of these eight instructions. A through H represent the names associated with eight different control bit patterns.

**TIME**: 1.25 microseconds.

---

#### RETURN OPERATION

<table>
<thead>
<tr>
<th>Address n</th>
<th>XEC&lt;sup&gt;-1&lt;/sup&gt;</th>
<th>This instruction results in the execution of the instruction located at the current value of the Program counter plus 1 plus the contents of R11, which is the caller index value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address 1</td>
<td>JMP B</td>
<td>The JMP B table follows in consecutive Program Storage locations following SEC.</td>
</tr>
<tr>
<td>Address 2</td>
<td>JMP C</td>
<td></td>
</tr>
<tr>
<td>Address 3</td>
<td>JMP D</td>
<td></td>
</tr>
<tr>
<td>Address 4</td>
<td>JMP D</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 20**

#### SUBPROGRAM CALL

<table>
<thead>
<tr>
<th>Program Storage Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMIT 0, R11</td>
<td>Load AUX with 0 Caller # 1</td>
</tr>
<tr>
<td>XMIT 1, R11</td>
<td>Load AUX with 1 Caller # 2</td>
</tr>
<tr>
<td>XMIT 2, R11</td>
<td>Load AUX with 2 Caller # 3</td>
</tr>
<tr>
<td>XMIT 3, R11</td>
<td>Load AUX with 3 Caller # 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Machine Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>XEC *+1(R11)</td>
<td>Execute JMP located at current PC + 1 + (R11).</td>
</tr>
<tr>
<td>JMP 000141x</td>
<td>Return to Caller #1</td>
</tr>
<tr>
<td>JMP 001135x</td>
<td>Return to Caller #2</td>
</tr>
<tr>
<td>JMP 003262x</td>
<td>Return to Caller #3</td>
</tr>
<tr>
<td>JMP 003656x</td>
<td>Return to Caller #4</td>
</tr>
</tbody>
</table>

**Figure 21**

---

#### 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 positive, 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

PROCTAD16
MCL1B PROCEDURE TO ADD TWO 16 BIT NUMBERS
IN 2'S COMPLEMENT NOTATION AND CHECK FOR ARITHMETIC OVERFLOW.
DATA DECLARATIONS.
AL RIV 200,7,8
AH RIV 201,7,8
A1 RIV 201,0,1
BL RIV 202,7,6
BH RIV 203,7,8
B1 RIV 203,0,1
TAD  MOV  A1,AUX
    SEL A1
    ADD B1,R1
    RF: 0 IF 80TH = 0.
    R1 = 2 IF BOTH = 1.
    R1 = 1 IF BOTH DIFF.
    MOV  AL,AUX
    SEL BL
    ADD BL,BL
    MOVE OFV,AUX
    SEL AH
    ADD AH,AUX
    SEL BH
    ADD BH,BH
    ANSWER IN BH . . . BL
    ORG 4,258
    XEC +1(R1)
    JMP ZEROS
    JMP INBOUNDS
    JMP ONES
    ORG 4,32
    ZEROS NZT B1,OVERFLOW
    JMP INBOUNDS
    ORG 5,32
    ONES NZT B1,INBOUNDS
    OVERFLOW XMIT 100H,AUX
    MOVE AUX,BH
    SEL BL
    XMIT 0,8L
    INBOUNDS RTN
    END TAD16
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 straightforward 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

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 5.556
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
ORG 16.256
LOOP XMIT 1.AUX LOW ORDER BIT ?
AND R1.AUX
NZT AUX "* 2
JMP SHIFT
MOVE R2.AUX YES ADD MULTIPICAND.
ADD R3.R3
SHIFT XMIT 177H.AUX SHIFT PARTIAL
AND R1(1).R1 PRODUCT RIGHT.
XMIT 200H.AUX
XMIT R3(1).AUX
XOR R1.R1
XMIT 177H.AUX
AND R3(1).R3
XMIT 1.AUX
ADD R4.R4
NZT R4 LOOP DONE?
ORG 9.216
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 R4.R4
ENDD RTN
END MUL8X8
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

Figure 5
M16X16 PROGRAM LISTING
PROC M16X16
MCLIB PROCEDURE TO MULTIPLY TWO
16-BIT 2's COMPLEMENT NUMBERS.

LINK RY 200.7
SRESULT PW 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
XOR TAB(AUX)
NZT AUX,COMP1
JMP OP2
TAB XMIT 1,SRESULT
XMIT 0,SRESULT

COMP1 XMIT 37H,AUX
XOR R4,R4
XOR R3(7),AUX
XMIT 1,AUX
ADD R4,R4
MOVE OVF,AUX
ADD R3,R3
OP2 XMIT 1,AUX
AND R3(7),AUX
XOR SRESULT,SRESULT
NZT AUX,COMP2
JMP LOOP

COMP2 XMIT 37H,AUX
XOR R6,R6
XOR R5(7),AUX
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 29.32
LOOP XMIT 1,AUX
AND R4,AUX
NZT AUX,** 2
JMP SHIFT

Figure 6 (Cont'd)

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 R2(1),R3
XMIT 200H,AUX
AND R1(1),AUX
XOR R2,R2
XMIT 177H,AUX
AND R1(1),R1
XMIT 177H,AUX
REPLACE NEG. AND R3(1),R3
OPERANDS WITH THEIR 2's
COMPLEMENT.
XMIT 200H,AUX
AND R4(1),R4
XMIT 177H,AUX
REPLACE NEG. AND R3(1),R3
OPERANDS WITH THEIR 2's
COMPLEMENT.
XMIT 200H,AUX
AND R4(1),R4
XMIT 177H,AUX
NEGATE RESULT IF NECESSARY.
ORG 14.22
NZT SRESULT,ENDD
XMIT 200H,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
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
DIV16X8 PROGRAM LISTING

PROC DIV16X8
16-BIT BY 6-BIT 2's COMP DIVIDE WITH DIVIDEND IN R1,R2 AND DIVISOR IN R3

LINK RI,R200,7,8
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 R2,R2

START XMIT -1,R4
XMIT 377H,AUX NEGATIVE OF DIVISOR
XOR R3,R3
XMIT 1,AUX
ADD R5,R5
ORG 20,256
LOOP MOVE R6,AUX
ADD R1,R1 TRIAL SUBTRACT
ORG 8,256
NZT OVF,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 R6,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 PROGRAM LISTING

PROC D24X12

MCUB 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 201,0,8
SIGN RIV 201,7,1
COUNT RIV 201,8,5
SWITCH RIV 201,8,1

SEL LINK

ADD R2,R2

PROC

D24X12

MOV 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 R1,R1

ADD R1,R1

XMIT -13,COUNT

MOVE OVF,AUX

XMIT 1,AUX

ADD R3,R3

XMIT COUNT,COUNT

ADD R3,R3

NZT COUNT, LOOP

MOVE OVF,AUX

ORG 15,32

ADD R2,R2

END D24X12

START XMIT 377H,AUX

NEGATE DIVISOR.

XOR R4,R6

XMIT 360H,AUX

ADD R1,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
**DTOB PROGRAM LISTING**

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

START XMIT 0,R3 INIALIZATION.
XMIT 0,R2
XMIT 0,R1
XMIT 7,R11
MOVE R6,RYR

ONE MOVE DMYL,R4
ORG 25,256
LOOP1 XEC TABA(R1) COMPLEMENT R1.
MOVE R1,AUX
MOVE R6,RyR
ADD R6,R6 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 TAB8-1(R11)
ADD R4,R4
ORG 29,256
XEC TAB8-1(R11)
ADD OVF,AUX
ADD R3,R3
ORG 32,256
XEC TAB8-1(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 300H,AUX THOUSAND.
XMIT 140H,AUX HUNDRED.
XMIT 12H,AUX TEN.

TAB4 XMIT 102H,AUX MILLION.
XMIT 206H,AUX HUNDRED THOUSAND.
XMIT 47H,AUX TEN THOUSAND.
XMIT 5AUX THOUSAND.
XMIT 0,AUX HUNDRED.
XMIT 5,AUX TEN.

TAB5 XMIT 17H,AUX MILLION.
XMIT 1,AUX HUNDRED THOUSAND.
XMIT 6,AUX TEN THOUSAND.
XMIT 6,AUX THOUSAND.
XMIT 6,AUX HUNDRED.
XMIT 6,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: 41/2 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 461141638 to BCD)

Calls On Other Library Procedures: None
BTOD PROGRAM LISTING
MCLIB PROCEDURE TO CONVERT A 24-BIT POSITIVE BINARY NUMBER INTO 7 BCD DIGITS
LINK RIV 200,7,8
DMYH RIV 200,3,4
DMYH RIV 200,3,4
SEL LINK STORE RETURN LINK.
MOVE R11,LINK
START XMIT 7,R1
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 ENABLE PROPER W.S. BYTE
MOVE 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,R5 * 2
JMP DONE
ORG 40,256
LOOP2 XEC TAB3 -1(R5) STORE RETURN LINK.
ADD R4,R4 ADD R5,R5
ORG 46,256
XEC TAB4 -1(R5) RESTORE LINK.
ADD R4,R4 ADD R5,R5
ORG 46,256
XEC TAB5 -1(R5) ADD R11,AUX
ADD R5,R5 ADD R11,AUX
ORG 46,256
XEC TAB6 -1(R5) ADD R11,AUX
ADD R5,R5 ADD R11,AUX
ORG 46,256
RESTORE XEC TAB8 -1(R5) ADD R4,R4
ADD R5,R5 ADD R11,AUX
ORG 50,256
XEC TAB8 -1(R5) ADD R2,R2
ADD R5,R5 ADD R2,R2
NXT R11,R11
ORG 46,256
XEC TAB9 -1(R5) JMP LOOP1
ADD R5,R5 XMIT 1,AUX
XEC TABA *1(R1)
JMP LOOP2
KEEP MOVE R4,DMYR MOV R11,LINK
RTN

Figure 14

<table>
<thead>
<tr>
<th>XMIT</th>
<th>ADD DMYL,DMYL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABB</td>
<td>ADD DMYH,DMYH</td>
</tr>
<tr>
<td>TABC</td>
<td>XMIT 0,DMYL</td>
</tr>
<tr>
<td>XMIT</td>
<td>XMIT 0,DMYH</td>
</tr>
</tbody>
</table>

Figure 14 (Cont'd)

Figure 14

Si!lDotiC!i

101
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 scratch pad 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

SORT PROGRAM LISTING

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 mantissa. If the exponent of an operand is found in memory location N, then the high order byte of the mantissa must be in location N + 1. The low order byte must be in location N + 2. Both the exponent and the mantissa 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 Functions:
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 operand (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.

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](image-url)
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 TRANSMITTED TO THE IVR REG. THUS THE ADDRESSES ASSIGNED TO THESE VARIABLES ARE NOT IMPORTANT.

SEL LINK SAVE FPADD RETURN LINK.

MOVE R1,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, =Z
JMP ADD 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 THAN X2.
ADD R6,R5 MAKE R6 POSITIVE.
XOR R6,R6
XMIT -16,AUX DO X1 AND X2 DIFFER BY MORE THAN 15?
ADD R6,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

ADD XMIT 1,AUX
ADD R1,R5 R5 POINTS TO HOB OF OP1.
ADD R2,R6 R6 POINTS TO HOB OF OP2.
CALL TADSF

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 occurrences 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 201,1,2
H2 RIV 201,2,3
H4 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. THIS IS NOT IMPORTANT WHAT ADDRESS THEY ARE ASSIGNED 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
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 036037.

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 PROGRAM LISTING

PROC TAD16F
MCLLB 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 200,8,8

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.
XMIT 1,AUX R4 = 2 IF BOTH NEG.
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 OP3.
MOVE OVF,AUX
MOVE R6,IVR ENABLE HOB OF OP2.
ADD DMY,AUX
MOVE R5,IVR ENABLE HOB OF OP1.
ADD DMY,DMY ANSWER IN OP2.

XEC "**1(R4)
JMP ZEROS
JMP INBOUNDS
JMP ONES
ORG 16,32

ZEROS N2T SIGN,OVERFLOW
JMP INBOUNDS
ONES N2T 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

INBOUNDS RTN
END TAD16F

Figure 22
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

<table>
<thead>
<tr>
<th>F,NA PACKAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UD7 1</td>
</tr>
<tr>
<td>UD6 2</td>
</tr>
<tr>
<td>UD5 3</td>
</tr>
<tr>
<td>UD4 4</td>
</tr>
<tr>
<td>UD3 5</td>
</tr>
<tr>
<td>UD2 6</td>
</tr>
<tr>
<td>UD1 7</td>
</tr>
<tr>
<td>UDO 8</td>
</tr>
<tr>
<td>BOC 9</td>
</tr>
<tr>
<td>BOC 10</td>
</tr>
<tr>
<td>ME 11</td>
</tr>
<tr>
<td>GND 12</td>
</tr>
<tr>
<td>Vcc 24</td>
</tr>
<tr>
<td>Vref 23</td>
</tr>
<tr>
<td>Vref 22</td>
</tr>
<tr>
<td>Vref 21</td>
</tr>
<tr>
<td>Vref 20</td>
</tr>
<tr>
<td>Vref 19</td>
</tr>
<tr>
<td>Vref 18</td>
</tr>
<tr>
<td>Vref 17</td>
</tr>
<tr>
<td>Vref 16</td>
</tr>
<tr>
<td>Vref 15</td>
</tr>
<tr>
<td>Vref 14</td>
</tr>
<tr>
<td>Vref 13</td>
</tr>
<tr>
<td>MCLK 11</td>
</tr>
</tbody>
</table>

BLOCK DIAGRAM
8-BIT LATCHED BIDIRECTIONAL I/O PORT

PIN DESIGNATION

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

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.

<table>
<thead>
<tr>
<th>BIC</th>
<th>BOC</th>
<th>MCLK</th>
<th>USER DATA BUS FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>L</td>
<td>X</td>
<td>Output Data</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>H</td>
<td>Input Data</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>X</td>
<td>Inactive</td>
</tr>
</tbody>
</table>

Table 1 USER PORT CONTROL FUNCTION

<table>
<thead>
<tr>
<th>ME</th>
<th>RC</th>
<th>WC</th>
<th>MCLK</th>
<th>BIC</th>
<th>MICROPROCESSOR BUS FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>Output Data</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>Input Data</td>
</tr>
<tr>
<td>X</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>Inactive</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>L</td>
<td>Inactive</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Inactive</td>
</tr>
</tbody>
</table>

Table 2 MICROPROCESSOR PORT CONTROL FUNCTION
### DC ELECTRICAL CHARACTERISTICS

$V_{CC} = 5\text{V }\pm 5\%, \ 0^\circ\text{C} \leq T_A \leq 70^\circ\text{C}$ unless otherwise specified.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$ High</td>
<td></td>
<td>2.0</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL}$ Low</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{IC}$ Clamp</td>
<td></td>
<td>.8</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}$ High</td>
<td>$V_{CC} = 5.25\text{V}$</td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$ Low</td>
<td></td>
<td>-1</td>
<td>V</td>
</tr>
<tr>
<td>Input current$^1$</td>
<td></td>
<td>&lt;10</td>
<td>$\mu\text{A}$</td>
</tr>
<tr>
<td>$I_{IH}$ High</td>
<td>$V_{CC} = 5.25\text{V}$</td>
<td>100</td>
<td>$\mu\text{A}$</td>
</tr>
<tr>
<td>$I_{IL}$ Low</td>
<td>$V_{IL} = .5\text{V}$</td>
<td>-350</td>
<td>$\mu\text{A}$</td>
</tr>
<tr>
<td>Output current$^2$</td>
<td>$V_{CC} = 4.75\text{V}$</td>
<td>-550</td>
<td>$\mu\text{A}$</td>
</tr>
<tr>
<td>$I_{OS}$ UD bus</td>
<td>$V_{CC} = 5.25\text{V}$</td>
<td>100</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{OS}$ IV bus</td>
<td></td>
<td>150</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{CC}$ VCC supply current</td>
<td>$V_{CC} = 5.25\text{V}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.

### LOAD CIRCUIT FOR TRI-STATE OUTPUTS

![Load Circuit Diagram]

**NOTE:** $C_L$ includes fixture capacitance.

### INPUT WAVEFORM

![Input Waveform Diagram]

### CLOCK PULSE WIDTH

![Clock Pulse Width Diagram]

### DATA DELAY TIMES

**Clock Referenced**

![Data Delay Times Diagram]
### AC Electrical Characteristics

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>INPUT</th>
<th>TEST CONDITION</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_PD ) User data delay (^1)</td>
<td>UDX, MCLK</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 25 Typ 45 Max 61</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OE} ) User output enable</td>
<td>BOC</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 18 Typ 26 Max 47</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OD} ) User output disable</td>
<td>BIC</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 18 Typ 26 Max 35</td>
<td>ns</td>
</tr>
<tr>
<td>( t_PD ) ( \mu P ) data delay (^1)</td>
<td>IVBX, MCLK</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 38 Typ 53 Max 61</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OE} ) ( \mu P ) output enable</td>
<td>ME, RC, WC</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 14 Typ 19 Max 25</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OD} ) ( \mu P ) output disable</td>
<td>ME, RC, WC</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 13 Typ 17 Max 32</td>
<td>ns</td>
</tr>
<tr>
<td>( t_W ) Minimum pulse width</td>
<td>MCLK</td>
<td></td>
<td>Min 40</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{SETUP} ) Minimum setup time (^2)</td>
<td>UDX(^3), BIC, IVX, ME, RC, WC</td>
<td>( C_L = 50\text{pF} )</td>
<td>Min 25 Typ 55 Max 30</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{HOLD} ) Minimum hold time (^2)</td>
<td>ME, RC, WC</td>
<td></td>
<td>Min 5 Typ 10 Max 5</td>
<td>ns</td>
</tr>
</tbody>
</table>

**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. Setup 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

- MCLK
- BIC
- DATA AND CONTROL
- \( 1.5\text{V} \) (Low Level)
- \( 5.5\text{V} \) (High Level)

#### OUTPUT ENABLE AND DISABLE TIMES

- CONTROL OUTPUT
- DATA OUTPUT 1
- DATA OUTPUT 2
- LOW LEVEL ENABLING
- HIGH LEVEL ENABLING
- \( V_{OL} \)
- \( V_{OH} \)

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 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:

NBTYY-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

BLOCK DIAGRAM
**PIN DESCRIPTION**

<table>
<thead>
<tr>
<th>PIN</th>
<th>SYMBOL</th>
<th>NAME AND FUNCTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>UD0-UD7</td>
<td>User Data I/O Lines. Bidirectional data lines to communicate with user's equipment. Either tri-state or open collector outputs are available.</td>
<td>Active high</td>
</tr>
<tr>
<td>16-23</td>
<td>IV0-IV7</td>
<td>Microprocessor Bus. Bidirectional data lines to communicate with controlling digital system (microprocessor).</td>
<td>Active low</td>
</tr>
<tr>
<td>10</td>
<td>BIC</td>
<td>Input Control. User input to control writing into the I/O Port from the user data lines.</td>
<td>Active low</td>
</tr>
<tr>
<td>9</td>
<td>BOC</td>
<td>Output Control. User input to control reading from the I/O Port onto the user data lines.</td>
<td>Active low</td>
</tr>
<tr>
<td>11</td>
<td>ME</td>
<td>Master Enable. System input to enable or disable all other system inputs and outputs. It has no effect on user inputs and outputs.</td>
<td>Active low</td>
</tr>
<tr>
<td>15</td>
<td>WC</td>
<td>Write Command. When WC is high and SC is low, I/O Port, if selected, stores contents of IV0-IV7 as data.</td>
<td>Active high</td>
</tr>
<tr>
<td>14</td>
<td>SC</td>
<td>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.</td>
<td>Active high</td>
</tr>
<tr>
<td>13</td>
<td>MCLK</td>
<td>Master Clock. Input to strobe data into the latches. See function tables for details.</td>
<td>Active high</td>
</tr>
<tr>
<td>12</td>
<td>GND</td>
<td>Ground.</td>
<td></td>
</tr>
</tbody>
</table>

**USER DATA BUS FUNCTION**

<table>
<thead>
<tr>
<th>BIC</th>
<th>BOC</th>
<th>MCLK</th>
<th>8T32, 8T33</th>
<th>8T35, 8T36</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>L</td>
<td>X</td>
<td>Output Data</td>
<td>Output Data</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>H</td>
<td>Input Data</td>
<td>Input Data</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>L</td>
<td>Inactive</td>
<td>Inactive</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>X</td>
<td>Inactive</td>
<td>Inactive</td>
</tr>
</tbody>
</table>

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

Table 1  USER PORT CONTROL FUNCTION

**MICROPROCESSOR BUS CONTROL**

As is shown in Table 2, the activity of the microprocessor port is controlled by the BIC and BOC inputs as well as the status 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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>INPUT</th>
<th>TEST CONDITION</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min</td>
<td>Typ</td>
</tr>
<tr>
<td>tPD User data delay</td>
<td>UDX</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>25</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>MCLK*</td>
<td></td>
<td>45</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>BIC†</td>
<td></td>
<td>40</td>
<td>55</td>
</tr>
<tr>
<td>tOE User output enable</td>
<td>BOC</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>tOD User output disable</td>
<td>BIC</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>BOC</td>
<td></td>
<td>28</td>
<td>33</td>
</tr>
<tr>
<td>tPD μP data delay</td>
<td>IVBX</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>38</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>MCLK</td>
<td></td>
<td>48</td>
<td>61</td>
</tr>
<tr>
<td>tOE μP output enable</td>
<td>ME</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>SC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tOD μP output disable</td>
<td>ME</td>
<td>C&lt;sub&gt;L&lt;/sub&gt; = 50pF</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>SC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tW Minimum pulse width</td>
<td>MCLK</td>
<td></td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIC†</td>
<td></td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UDX†</td>
<td></td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIC</td>
<td></td>
<td>25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IVX</td>
<td></td>
<td>55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SC</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WC</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>tSETUP Minimum setup time</td>
<td>(Note 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tHOLD Minimum hold time</td>
<td>(Note 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 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 operation. 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

$0^\circ C \leq T_A \leq 70^\circ C$, $V_{CC} = 5V \pm 5$

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$ High-level input voltage</td>
<td>$I_I = -5mA$</td>
<td>Min 2.0 Typ 5.5 Max V</td>
<td></td>
</tr>
<tr>
<td>$V_{IL}$ Low-level input voltage</td>
<td></td>
<td>Min -1.0 Typ 8 Max V</td>
<td></td>
</tr>
<tr>
<td>$V_{CL}$ Input clamp voltage</td>
<td>$V_{IH} = 5.25V$</td>
<td>&lt;10 100 $\mu A$</td>
<td></td>
</tr>
<tr>
<td>$I_{IH}$ High-level input current$^1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$ Low level input current$^1$</td>
<td>$V_{IL} = 5V$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OL}$ Low-level output voltage</td>
<td>$V_{CC} = 4.75V$</td>
<td>0.55 $V$</td>
<td></td>
</tr>
<tr>
<td>$V_{OH}$ High-level output voltage</td>
<td>$I_{OL} = 16mA$</td>
<td>2.4 $V$</td>
<td></td>
</tr>
<tr>
<td>$I_{OS}$ Short-circuit output current$^2$</td>
<td></td>
<td>10 mA</td>
<td></td>
</tr>
<tr>
<td>$I_{CC}$ Supply current</td>
<td></td>
<td>100 150 $mA$</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

### TEST LOAD CIRCUIT (OPEN COLLECTOR OUTPUTS)

**INPUT WAVEFORM**

- $V_{IH} = 5.5V$
- $V_{IL} = -1.5V$
- $t_f < 5\mu s$

**VOLTAGE WAVEFORMS**

- $V_{CC} = 5.25V$

### TEST LOAD CIRCUIT (TRI-STATE OUTPUTS)

**CLOCK PULSE WIDTH**

- $1.5V$
- Width $t_w$

### Absolute Maximum Ratings:

- Supply voltage $3 \ldots \ldots 7V$
- Input voltage $3 \ldots \ldots 5.5V$
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 = VCC, SC = WC = MCLK = GND). Leave all Microprocessor Bus I/O pins open.
2. Raise VCC to 7.75V ± .25V.
3. After VCC 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 VCC 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 VCC 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:

1. Set VCC and all control inputs to 0V. (VCC = BIC = BOC = ME = WC = SC = MCLK = GND). 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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCP</td>
<td>Programming supply voltage Address</td>
<td>7.5</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ICCP</td>
<td>Programming supply current Address</td>
<td>17.5</td>
<td>16.0</td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>13.5</td>
<td>14.0</td>
</tr>
<tr>
<td></td>
<td>Max time VCCP &gt; 5.25V Address</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming voltage Address</td>
<td>17.5</td>
<td>16.0</td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>13.5</td>
<td>14.0</td>
</tr>
<tr>
<td></td>
<td>Programming current Address</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming pulse rise time Address</td>
<td>.1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Protect</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Programming pulse width</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

NOTE

1. If all programming can be done in less than 1 second, VCC 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.
ADDRESSABLE BUS EXPANDER

PRELIMINARY SPECIFICATION

8T39-1,XL

DESCRIPTION
The Bus Expander is specifically designed to increase the I/O capability of the 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 I 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 I I 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.

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 D1 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 1/0 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

<table>
<thead>
<tr>
<th>PIN NO.</th>
<th>SYMBOL</th>
<th>NAME AND FUNCTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-7,9,10</td>
<td>D00-D07</td>
<td>I/O port data bus</td>
<td>Active low, three-state</td>
</tr>
<tr>
<td>11</td>
<td>WC(O/X)</td>
<td>Write command output</td>
<td>Active high</td>
</tr>
<tr>
<td>12</td>
<td>SC(O/X)</td>
<td>Select command output</td>
<td>Active high</td>
</tr>
<tr>
<td>13</td>
<td>MCLK(O/X)</td>
<td>Master clock output</td>
<td>Active high</td>
</tr>
<tr>
<td>14</td>
<td>ME(O/X)</td>
<td>Master enable output</td>
<td>Active high</td>
</tr>
<tr>
<td>15</td>
<td>ME(IN)</td>
<td>Master enable input</td>
<td>Active high</td>
</tr>
<tr>
<td>16</td>
<td>MCLK(IN)</td>
<td>Master clock input</td>
<td>Active high</td>
</tr>
<tr>
<td>17</td>
<td>SC(IN)</td>
<td>Select command output</td>
<td>Active high</td>
</tr>
<tr>
<td>18</td>
<td>WC(IN)</td>
<td>Write command output</td>
<td>Active high</td>
</tr>
<tr>
<td>19,20,22-27</td>
<td>D10-D17</td>
<td>Microprocessor data bus</td>
<td>Active low, three-state</td>
</tr>
<tr>
<td>1,8,21</td>
<td>GND</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>VCC</td>
<td>+5 volt supply</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 8T39 ADDRESSING SUMMARY
## TRUTH TABLE

<table>
<thead>
<tr>
<th>ME</th>
<th>SC</th>
<th>WC</th>
<th>MCLK</th>
<th>SELECT LATCH</th>
<th>DATA TRANSFER DIRECTION</th>
<th>ADDRESS* COMPARISON</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>Set</td>
<td>DI Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>Not set</td>
<td>DI Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>DI Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>DI Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>DI Bus → DO Bus</td>
<td>Yes</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>DI Bus → DO Bus</td>
<td>No</td>
</tr>
</tbody>
</table>

**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

[Diagram of the functional block diagram]

**NOTES**

Selection made during manufacture

X = Don't care
**ABSOLUTE MAXIMUM RATINGS**

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RATING</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>+7 Vdc</td>
<td></td>
</tr>
<tr>
<td>VIN</td>
<td>+5.5 Vdc</td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>+5.5 Vdc</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>0°C to +70°C</td>
<td></td>
</tr>
<tr>
<td>TSTG</td>
<td>-65 to +150°C</td>
<td></td>
</tr>
</tbody>
</table>

**ORDERING INFORMATION**

The Bus Expander is ordered by specifying the following part number: N8T39-XX P

- **P =** Ceramic Package
- **XL =** Epoxy Package
- Address Range As Determined From Table 1

**DC ELECTRICAL CHARACTERISTICS** \( V_{CC} = 5V \pm 5\% \), \( 0°C \leq T_A \leq 70°C \)

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Low</td>
<td>-5mA at ( V_{CC} ) min</td>
<td>2.0</td>
</tr>
<tr>
<td>VIH</td>
<td>High</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>VIC</td>
<td>Clamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Low</td>
<td>( V_{CC} = 4.75V )</td>
<td>2.4</td>
</tr>
<tr>
<td>VOH</td>
<td>High</td>
<td>( I_{OL} = 16mA )</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Low(^1)</td>
<td>( V_{IL} = 5.25V )</td>
<td>&lt; 10</td>
</tr>
<tr>
<td>IIH</td>
<td>High(^1)</td>
<td>( V_{IH} = 5.25V )</td>
<td>100</td>
</tr>
<tr>
<td>IOS</td>
<td>Short circuit output current</td>
<td>( V_{CC} = 4.75V )</td>
<td>-40</td>
</tr>
<tr>
<td>ICC</td>
<td>Supply current</td>
<td>( V_{CC} = 5.25V )</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TO</th>
<th>FROM</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpd</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (out) MCLK (out)</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>tpd</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (in) MCLK (in)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>toe</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (in) SC (in) WC (in)</td>
<td>28 56 ns</td>
<td></td>
</tr>
<tr>
<td>tod</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (in) SC (in) WC (in)</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>tsetup</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (in) MCLK (in) SC (in) WC (in)</td>
<td>54 ns</td>
<td></td>
</tr>
<tr>
<td>thold</td>
<td>DIX</td>
<td>DIX</td>
<td>DIX ME (in) MCLK (in) SC (in) WC (in)</td>
<td>3 ns</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**

1. Includes tri-state leakage.
2. Minimum clock width \( \approx 50\)ns.
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

VOLTAGE WAVEFORMS

PROPERGATION DELAY TO
THREE-STATE OUTPUTS

TEST LOAD CIRCUIT

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

<table>
<thead>
<tr>
<th>I, XL PACKAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND 1</td>
</tr>
<tr>
<td>DO7 2</td>
</tr>
<tr>
<td>DO6 3</td>
</tr>
<tr>
<td>DO5 4</td>
</tr>
<tr>
<td>DO4 5</td>
</tr>
<tr>
<td>DO3 6</td>
</tr>
<tr>
<td>DO2 7</td>
</tr>
<tr>
<td>GND 8</td>
</tr>
<tr>
<td>DO1 9</td>
</tr>
<tr>
<td>DO0 10</td>
</tr>
<tr>
<td>WC (OUT) 11</td>
</tr>
<tr>
<td>SC (OUT) 12</td>
</tr>
<tr>
<td>MCLK (OUT) 13</td>
</tr>
<tr>
<td>ME (OUT) 14</td>
</tr>
<tr>
<td>+5V supply</td>
</tr>
</tbody>
</table>

TRUTH TABLE

<table>
<thead>
<tr>
<th>ME</th>
<th>SC</th>
<th>WC</th>
<th>DATA TRANSFER DIRECTION</th>
<th>ADDRESS COMPARISON</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Di Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>Di Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>X</td>
<td>Di Bus → DO Bus</td>
<td>No</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>Di Bus → DO Bus</td>
<td>No</td>
</tr>
</tbody>
</table>

PIN DESIGNATION

<table>
<thead>
<tr>
<th>PIN NO.</th>
<th>SYMBOL</th>
<th>NAME &amp; FUNCTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-7,9,10</td>
<td>DO0-DO7</td>
<td>I/O port data bus</td>
<td>Active low, three-state</td>
</tr>
<tr>
<td>11</td>
<td>WC(OUT)</td>
<td>Write command output</td>
<td>Active high</td>
</tr>
<tr>
<td>12</td>
<td>SC(OUT)</td>
<td>Select command output</td>
<td>Active high</td>
</tr>
<tr>
<td>13</td>
<td>MCLK(OUT)</td>
<td>Master clock input</td>
<td>Active high</td>
</tr>
<tr>
<td>14</td>
<td>ME(OUT)</td>
<td>Master enable output</td>
<td>Active low</td>
</tr>
<tr>
<td>15</td>
<td>ME(IN)</td>
<td>Master enable input</td>
<td>Active low</td>
</tr>
<tr>
<td>16</td>
<td>MCLK(IN)</td>
<td>Master clock input</td>
<td>Active high</td>
</tr>
<tr>
<td>17</td>
<td>SC(IN)</td>
<td>Select command output</td>
<td>Active high</td>
</tr>
<tr>
<td>18</td>
<td>WC(IN)</td>
<td>Write command output</td>
<td>Active high</td>
</tr>
<tr>
<td>19,20,22-27</td>
<td>DIO-D17</td>
<td>Microprocessor data bus</td>
<td>Active low, three-state</td>
</tr>
<tr>
<td>1,8,21</td>
<td>GND</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Vcc</td>
<td>+5V supply</td>
<td></td>
</tr>
</tbody>
</table>
Preliminary Specification

Functional Block Diagram

Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RATING</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vcc</td>
<td>+7 Vdc</td>
<td></td>
</tr>
<tr>
<td>Vin</td>
<td>+5.5 Vdc</td>
<td></td>
</tr>
<tr>
<td>V0</td>
<td>+5.5 Vdc</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>0 to +70 °C</td>
<td></td>
</tr>
<tr>
<td>TSTG</td>
<td>-65 to +150 °C</td>
<td></td>
</tr>
</tbody>
</table>

Ac Electrical Characteristics

\[ Vcc = 5V \pm 5\% \], \[ 0°C \leq TA \leq 70°C \], \[ CL = 300pF \]

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TO</th>
<th>FROM</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpd</td>
<td>DOX</td>
<td>DIX</td>
<td>DIX</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>tpd Control</td>
<td>ME(OUT)</td>
<td>ME(IN)</td>
<td>MCLK(OUT)</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>tpd</td>
<td>SCI(OUT)</td>
<td>SCI(IN)</td>
<td>WC(OUT)</td>
<td>15 ns</td>
<td></td>
</tr>
<tr>
<td>tce</td>
<td>DIX</td>
<td>ME(IN)</td>
<td>SC(OUT)</td>
<td>28 ns</td>
<td></td>
</tr>
<tr>
<td>tcd</td>
<td>DIX</td>
<td>ME(IN)</td>
<td>WC(OUT)</td>
<td>15 ns</td>
<td></td>
</tr>
</tbody>
</table>
**TRANSPARENT BUS EXPANDER**

**PRELIMINARY SPECIFICATION**

**DC ELECTRICAL CHARACTERISTICS** $V_{CC} = 5V \pm 5\%$, $0^\circ C \leq T_A \leq 70^\circ C$

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$ Low</td>
<td></td>
<td>2.0</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH}$ High</td>
<td></td>
<td>.8</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IC}$ Clamp</td>
<td></td>
<td>-1</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$ Low</td>
<td>$V_{CC} = 4.75V$</td>
<td>.55</td>
<td>V</td>
</tr>
<tr>
<td>$I_{OL}$</td>
<td>$I_{OL} = 50mA$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{OH}$ High</td>
<td>$V_{IH} = 5.25V$</td>
<td>2.4</td>
<td>μA</td>
</tr>
<tr>
<td>$I_{IL}$ Low*</td>
<td>$V_{IL} = .5V$</td>
<td>-250</td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$ High*</td>
<td>$V_{IH} = 5.25V$</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>$I_{OS}$</td>
<td>$V_{CC} = 4.75V$</td>
<td>-40</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{CC}$ Supply current</td>
<td>$V_{CC} = 5.25V$</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

*Includes 3-State leakage.

**VOLTAGE WAVEFORMS**

**PROPORTION DELAY TO THREE-STATE OUTPUTS**

**PROPORTION DELAY TIMES**

**TEST LOAD CIRCUIT**

**TYPICAL APPLICATION**

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

All resistors values are typical and in ohms.

**NOTES**

1. $C_L$ includes probe and jig capacitance.
2. All diodes are 1N916 or 1N3064.
### BIPOLAR MEMORY SELECTION GUIDE

<table>
<thead>
<tr>
<th>DEVICE</th>
<th>ORGANIZATION</th>
<th>OUTPUT CIRCUIT</th>
<th>OUTPUT LOGIC</th>
<th>ACCESS TIME [ns]</th>
<th>TEMPERATURE RANGE</th>
<th>PACKAGE</th>
<th>NO. OF PINS</th>
<th>MAX. ICC [mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMS 10155</td>
<td>8X2</td>
<td>OE</td>
<td>—</td>
<td>13</td>
<td>C</td>
<td>F,N</td>
<td>18</td>
<td>140</td>
</tr>
<tr>
<td>SAMS 82S12</td>
<td>8X4</td>
<td>OC</td>
<td>T</td>
<td>40</td>
<td>C</td>
<td>F.N</td>
<td>24</td>
<td>160</td>
</tr>
<tr>
<td>SAMS 82S112</td>
<td>8X4</td>
<td>TS</td>
<td>T</td>
<td>40</td>
<td>C</td>
<td>F.N</td>
<td>24</td>
<td>160</td>
</tr>
<tr>
<td>RAMS 82S25</td>
<td>16X4</td>
<td>OC</td>
<td>B</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>105</td>
</tr>
<tr>
<td>3101A</td>
<td>16X4</td>
<td>OC</td>
<td>B</td>
<td>35</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>105</td>
</tr>
<tr>
<td>54/74S189</td>
<td>16X4</td>
<td>OC</td>
<td>T</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>105</td>
</tr>
<tr>
<td>54/74S189</td>
<td>16X4</td>
<td>TS</td>
<td>B</td>
<td>35</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>110</td>
</tr>
<tr>
<td>82S21</td>
<td>32X2</td>
<td>OC</td>
<td>T</td>
<td>50</td>
<td>C</td>
<td>F.N</td>
<td>16</td>
<td>130</td>
</tr>
<tr>
<td>82S16</td>
<td>256X1</td>
<td>TS</td>
<td>T</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>115</td>
</tr>
<tr>
<td>82S116</td>
<td>256X1</td>
<td>TS</td>
<td>T</td>
<td>40</td>
<td>C</td>
<td>F.N</td>
<td>16</td>
<td>115</td>
</tr>
<tr>
<td>82S116</td>
<td>256X1</td>
<td>OC</td>
<td>T</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>115</td>
</tr>
<tr>
<td>82S1117</td>
<td>256X1</td>
<td>OC</td>
<td>T</td>
<td>40</td>
<td>C</td>
<td>F.N</td>
<td>16</td>
<td>115</td>
</tr>
<tr>
<td>54/74S200</td>
<td>256X1</td>
<td>TS</td>
<td>B</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>130</td>
</tr>
<tr>
<td>54/74S201</td>
<td>256X1</td>
<td>TS</td>
<td>B</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>130</td>
</tr>
<tr>
<td>82S09</td>
<td>64X9</td>
<td>OC</td>
<td>T</td>
<td>45</td>
<td>M.C</td>
<td>I,N</td>
<td>28</td>
<td>190</td>
</tr>
<tr>
<td>82S10</td>
<td>1024X1</td>
<td>OC</td>
<td>B</td>
<td>45</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>82S110</td>
<td>1024X1</td>
<td>OC</td>
<td>B</td>
<td>35</td>
<td>C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>82S111</td>
<td>1024X1</td>
<td>TS</td>
<td>B</td>
<td>45</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>82S111</td>
<td>1024X1</td>
<td>TS</td>
<td>B</td>
<td>35</td>
<td>C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>93415A</td>
<td>1024X1</td>
<td>OC</td>
<td>B</td>
<td>45</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>93425A</td>
<td>1024X1</td>
<td>TS</td>
<td>B</td>
<td>45</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>82S208*</td>
<td>256X9</td>
<td>TS</td>
<td>B</td>
<td>60</td>
<td>C</td>
<td>F</td>
<td>22</td>
<td>185</td>
</tr>
<tr>
<td>82S210*</td>
<td>256X9</td>
<td>TS</td>
<td>B</td>
<td>60</td>
<td>C</td>
<td>F.N</td>
<td>24</td>
<td>185</td>
</tr>
<tr>
<td>82S400*</td>
<td>4096X1</td>
<td>OC</td>
<td>B</td>
<td>70</td>
<td>C</td>
<td>I</td>
<td>18</td>
<td>155</td>
</tr>
<tr>
<td>82S401*</td>
<td>4096X1</td>
<td>TS</td>
<td>B</td>
<td>70</td>
<td>C</td>
<td>I</td>
<td>18</td>
<td>155</td>
</tr>
<tr>
<td>ROMS 82S226</td>
<td>256X4</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>120</td>
</tr>
<tr>
<td>82S229</td>
<td>256X4</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>120</td>
</tr>
<tr>
<td>82S214</td>
<td>256X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>175</td>
</tr>
<tr>
<td>82S230</td>
<td>512X4</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>140</td>
</tr>
<tr>
<td>82S231</td>
<td>512X4</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M.C</td>
<td>F.N</td>
<td>16</td>
<td>140</td>
</tr>
<tr>
<td>82S215</td>
<td>512X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>175</td>
</tr>
<tr>
<td>82S240</td>
<td>512X8</td>
<td>OC</td>
<td>—</td>
<td>60</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>175</td>
</tr>
<tr>
<td>82S241</td>
<td>512X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>175</td>
</tr>
<tr>
<td>82S28</td>
<td>1024X4</td>
<td>TTL</td>
<td>—</td>
<td>50</td>
<td>C</td>
<td>F</td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>82S280</td>
<td>1024X8</td>
<td>OC</td>
<td>—</td>
<td>70</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>140</td>
</tr>
<tr>
<td>82S281</td>
<td>1024X8</td>
<td>TS</td>
<td>—</td>
<td>70</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>140</td>
</tr>
<tr>
<td>82S290</td>
<td>2048X8</td>
<td>OC</td>
<td>—</td>
<td>80</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>170</td>
</tr>
<tr>
<td>82S291</td>
<td>2048X8</td>
<td>TS</td>
<td>—</td>
<td>80</td>
<td>M.C</td>
<td>F.N</td>
<td>24</td>
<td>170</td>
</tr>
</tbody>
</table>

*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 (-5°C to +125°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)

<table>
<thead>
<tr>
<th>DEVICE</th>
<th>ORGANIZATION</th>
<th>OUTPUT CIRCUIT</th>
<th>OUTPUT LOGIC</th>
<th>ACCESS TIME (ns)</th>
<th>TEMPERATURE RANGE</th>
<th>PACKAGE</th>
<th>NO. OF PINS</th>
<th>MAX. ICC (mA)</th>
<th>EQUIVALENT ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82S23</td>
<td>32X8</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>77</td>
<td>—</td>
</tr>
<tr>
<td>82S123</td>
<td>32X8</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>77</td>
<td>—</td>
</tr>
<tr>
<td>10139</td>
<td>32X8</td>
<td>OE</td>
<td>—</td>
<td>15</td>
<td>C</td>
<td>F,N</td>
<td>16</td>
<td>145</td>
<td>—</td>
</tr>
<tr>
<td>82S27</td>
<td>256X4</td>
<td>OC</td>
<td>—</td>
<td>40</td>
<td>C</td>
<td>F</td>
<td>16</td>
<td>140</td>
<td>—</td>
</tr>
<tr>
<td>82S126</td>
<td>256X4</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>120</td>
<td>82S226</td>
</tr>
<tr>
<td>82S129</td>
<td>256X4</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>120</td>
<td>82S229</td>
</tr>
<tr>
<td>10149</td>
<td>256X4</td>
<td>OE</td>
<td>—</td>
<td>20</td>
<td>C</td>
<td>F</td>
<td>16</td>
<td>150</td>
<td>—</td>
</tr>
<tr>
<td>82S114</td>
<td>256X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S214</td>
</tr>
<tr>
<td>82S130</td>
<td>512X4</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>140</td>
<td>82S230</td>
</tr>
<tr>
<td>82S131</td>
<td>512X4</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>F,N</td>
<td>16</td>
<td>140</td>
<td>82S231</td>
</tr>
<tr>
<td>82S115</td>
<td>512X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S215</td>
</tr>
<tr>
<td>82S140</td>
<td>512X8</td>
<td>OC</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S240</td>
</tr>
<tr>
<td>82S141</td>
<td>512X8</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S241</td>
</tr>
<tr>
<td>82S136</td>
<td>1024X4</td>
<td>OC</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>18</td>
<td>140</td>
<td>—</td>
</tr>
<tr>
<td>82S137</td>
<td>1024X4</td>
<td>TS</td>
<td>—</td>
<td>60</td>
<td>M,C</td>
<td>F,N</td>
<td>18</td>
<td>140</td>
<td>—</td>
</tr>
<tr>
<td>82S180</td>
<td>1024X8</td>
<td>OC</td>
<td>—</td>
<td>70</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S280</td>
</tr>
<tr>
<td>82S181</td>
<td>1024X8</td>
<td>TS</td>
<td>—</td>
<td>70</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>82S281</td>
</tr>
<tr>
<td>82S270</td>
<td>1024X8</td>
<td>TS</td>
<td>—</td>
<td>70</td>
<td>M,C</td>
<td>F,N</td>
<td>24</td>
<td>175</td>
<td>—</td>
</tr>
<tr>
<td>82S184</td>
<td>2048X4</td>
<td>OC</td>
<td>—</td>
<td>100</td>
<td>M,C</td>
<td>I</td>
<td>18</td>
<td>120</td>
<td>—</td>
</tr>
<tr>
<td>82S185</td>
<td>2048X4</td>
<td>TS</td>
<td>—</td>
<td>100</td>
<td>M,C</td>
<td>I</td>
<td>18</td>
<td>120</td>
<td>—</td>
</tr>
<tr>
<td>82S190</td>
<td>2048X8</td>
<td>OC</td>
<td>—</td>
<td>70</td>
<td>M,C</td>
<td>I</td>
<td>24</td>
<td>175</td>
<td>82S290</td>
</tr>
<tr>
<td>82S191</td>
<td>2048X8</td>
<td>TS</td>
<td>—</td>
<td>70</td>
<td>M,C</td>
<td>I</td>
<td>24</td>
<td>175</td>
<td>82S291</td>
</tr>
<tr>
<td>FPLAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82S100</td>
<td>16X48X8</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
<tr>
<td>82S101</td>
<td>16X48X8</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
<tr>
<td>PLAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82S200</td>
<td>16X48X8</td>
<td>TS</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
<tr>
<td>82S201</td>
<td>16X48X8</td>
<td>OC</td>
<td>—</td>
<td>50</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
<tr>
<td>FPAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82S102</td>
<td>16X9</td>
<td>OC</td>
<td>—</td>
<td>30</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
<tr>
<td>82S103</td>
<td>16X9</td>
<td>TS</td>
<td>—</td>
<td>30</td>
<td>M,C</td>
<td>I,N</td>
<td>28</td>
<td>170</td>
<td>—</td>
</tr>
</tbody>
</table>

*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

BLOCK DIAGRAM

FE1 = (13), FE2 = (11), 5
VCC = (24), GND = (12), ( ) = Pin Number
### Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RATING</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_CC</td>
<td>+7</td>
<td>Vdc</td>
</tr>
<tr>
<td>V_IN</td>
<td>+5.5</td>
<td>Vdc</td>
</tr>
<tr>
<td>TA</td>
<td>-65 to +150</td>
<td>°C</td>
</tr>
<tr>
<td>T_STG</td>
<td>-55 to +125</td>
<td>°C</td>
</tr>
<tr>
<td></td>
<td>0 to +75</td>
<td>°C</td>
</tr>
<tr>
<td></td>
<td>N82S114/115</td>
<td>°C</td>
</tr>
<tr>
<td></td>
<td>S82S114/115</td>
<td>°C</td>
</tr>
</tbody>
</table>

### DC Electrical Characteristics

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>N82S114/115</th>
<th>S82S114/115</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>V_L</td>
<td>Low</td>
<td>2.0</td>
<td>.85</td>
<td>.8</td>
</tr>
<tr>
<td>V_H</td>
<td>High</td>
<td>-0.8</td>
<td>-1.2</td>
<td>-0.8</td>
</tr>
<tr>
<td>I_C</td>
<td>Clamp</td>
<td>I_{IN} = -18mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_O</td>
<td>Low</td>
<td>0.4</td>
<td>0.45</td>
<td>0.4</td>
</tr>
<tr>
<td>V_H</td>
<td>High</td>
<td>2.7</td>
<td>3.3</td>
<td>2.4</td>
</tr>
<tr>
<td>I_L</td>
<td>Low</td>
<td>V_{IN} = 0.45V</td>
<td>-100</td>
<td></td>
</tr>
<tr>
<td>I_H</td>
<td>High</td>
<td>V_{IN} = 5.5V</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>I_O(OFF)</td>
<td>Hi-Z state</td>
<td>CE_1 = Low, CE_2 = High, I_{OUT} = -2mA, High stored</td>
<td>-20</td>
<td>-40</td>
</tr>
<tr>
<td>I_O(S)</td>
<td>Short circuit2</td>
<td>CE_1 = High or CE_2 = 0, V_O = 5.5V</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>I_CC</td>
<td>V_CC supply current</td>
<td>130</td>
<td>175</td>
<td>130</td>
</tr>
<tr>
<td>C_IN</td>
<td>Capacitance</td>
<td>V_{CC} = 5.0V, V_{IN} = 2.0V</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>C_OUT</td>
<td>Input</td>
<td>V_{CC} = 5.0V, V_O = 2.0V</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Output</td>
<td>CE_1 = High or CE_2 = 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### AC Electrical Characteristics

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TO</th>
<th>FROM</th>
<th>TEST CONDITIONS</th>
<th>N82S114/115</th>
<th>S82S114/115</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_A</td>
<td>Out</td>
<td>Out</td>
<td>Address</td>
<td>Latched or transparent read</td>
<td>35</td>
<td>60</td>
</tr>
<tr>
<td>T_CE</td>
<td>Out</td>
<td>Out</td>
<td>Chip enable</td>
<td>Latched or transparent read</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>T_CD</td>
<td>Out</td>
<td>Out</td>
<td>Chip disable</td>
<td>Latched or transparent read</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>T_CDS</td>
<td>Out</td>
<td>Out</td>
<td>Chip enable</td>
<td>Latched read only</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>T_CDH</td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_ADH</td>
<td>0</td>
<td>-10</td>
<td>Address</td>
<td>Latched read only</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>T_SW</td>
<td>30</td>
<td>20</td>
<td>Strobe</td>
<td>Latched read only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_L</td>
<td>60</td>
<td>35</td>
<td>Strobe</td>
<td>Latched read only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_DL</td>
<td>30</td>
<td>35</td>
<td>Strobe</td>
<td>Latched read only</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES** on following page.
NOTES
1. Typical values are at Vcc = +5.0V and TA = +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 TA nanoseconds after the address has changed and Tce nanoseconds after the output circuit is enabled. Tcc 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.
PROGRAMMING SYSTEMS SPECIFICATIONS (Testing of these limits may cause programming of device.) $T_A = +25^\circ C$

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>LIMITS</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Typ</td>
</tr>
<tr>
<td>$V_{CCP}$</td>
<td>Power supply voltage</td>
<td>4.75</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>To program(^1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$I_{CCP} = 200 \pm 25\text{mA}$,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transient or steady state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{CCH}$</td>
<td>Verify limit</td>
<td>5.3</td>
<td>5.5</td>
</tr>
<tr>
<td>$V_{CCL}$</td>
<td>Upper</td>
<td>4.3</td>
<td>4.5</td>
</tr>
<tr>
<td>$V_S$</td>
<td>Lower</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>$I_{CCP}$</td>
<td>Programming supply current</td>
<td>175</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>$V_{CCP} = +5.0 \pm .2\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>Input voltage</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>High</td>
<td>2.4</td>
<td>4.8</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input current (FE(_1) &amp; FE(_2) only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>$V_{IL} = +0.45\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>$V_{IH} = +5.5\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input current (except FE(_1) &amp; FE(_2))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>$V_{IL} = +0.45\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>$V_{IH} = +5.5\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output programming voltage(^3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$I_{OUT} = 200 \pm 20\text{mA}$,</td>
<td>16.0</td>
<td>17.0</td>
</tr>
<tr>
<td></td>
<td>Transient or steady state</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$V_{OUT} = +17 \pm 1\text{V}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{OUT}$</td>
<td>Output programming current</td>
<td>180</td>
<td>200</td>
</tr>
<tr>
<td>$T_{R}$</td>
<td>Output pulse rise time</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>$t_{p}$</td>
<td>FE(_2) programming pulse width</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>$T_{0}$</td>
<td>Pulse sequence delay</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>$T_{PA}$</td>
<td>Programming time</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$T_{PS}$</td>
<td>Programming pause</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{PR}$</td>
<td>Programming duty cycle(^4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES
1. Bypass $V_{EE}$ to GND with a 0.01\(\mu\)F capacitor to reduce voltage spikes.
2. $V_S$ 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 ensure that the voltage output 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 ($V_{EE} = 0\text{V}$) of 3\(\mu\)s.

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 10k\(\Omega\) resistor to $V_{CC}$.
3. Set $CE_1$ to logical low, and $CE_2$ to logical high (TTL levels).
4. Set Strobe to logical 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 10\(\mu\)s delay, apply to $FE_1$ (pin 13) a voltage source of $+5.0 \pm 0.5\text{V}$, with 10mA sourcing current capability.
3. After 10\(\mu\)s delay, apply a voltage source of $+17.0 \pm 1.0\text{V}$ to the output to be programmed. The source must have a current limit of 200mA. Program on output at the time.
4. After 10\(\mu\)s delay, raise $FE_2$ (pin 11) from 0V to $+5.0 \pm 0.5\text{V}$ for a period of 1ms, and then return to 0V. Pulse source must have a 10mA sourcing current capability.
5. After 10\(\mu\)s delay, remove $+17.0\text{V}$ supply from programmed output.
6. To verify programming, after 10\(\mu\)s delay, return $FE_1$ to 0V. Raise $V_{CC}$ to $V_{CCH} = +5.5 \pm .2\text{V}$. The programmed output should remain in the high state. Again, lower $V_{CC}$ to $V_{CCL} = +4.5 \pm .2\text{V}$, 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.

.same addre
TYPICAL PROGRAMMING SEQUENCE

*Programming verification at both high and low VCC margins is optional. For convenience, verification can also be executed at the operating VCC 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 25\(\mu\)A for a high level, and -100\(\mu\)A 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 N82S16/116/1717, 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: -100\(\mu\)A
- 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

<table>
<thead>
<tr>
<th>MODE</th>
<th>CE*</th>
<th>WE</th>
<th>D_IN</th>
<th>D_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>Stored data</td>
</tr>
<tr>
<td>Write &quot;0&quot;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Write &quot;1&quot;</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Disabled</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>High-Z</td>
</tr>
</tbody>
</table>

**F = Cerdip**

**N = Plastic**

**Truth Table**

**Block Diagram**
### ABSOLUTE MAXIMUM RATINGS

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RATING</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{CC}$</td>
<td>+7</td>
<td>Vdc</td>
</tr>
<tr>
<td>$V_{IN}$</td>
<td>+5.5</td>
<td>Vdc</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>+5.5</td>
<td>Vdc</td>
</tr>
<tr>
<td>$V_O$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_A$</td>
<td>-55 to +125</td>
<td>°C</td>
</tr>
<tr>
<td>$T_{STG}$</td>
<td>-65 to +150</td>
<td></td>
</tr>
</tbody>
</table>

### DC ELECTRICAL CHARACTERISTICS

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

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

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>N82S16/17/116/117</th>
<th>S82S16/17</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$</td>
<td>High</td>
<td>$V_{CC} = \text{Max}$</td>
<td>2.0</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>Low</td>
<td>$V_{CC} = \text{Min}$</td>
<td>-1.0</td>
<td></td>
</tr>
<tr>
<td>$V_{IC}$</td>
<td>Clamp$^3$</td>
<td>$V_{CC} = \text{Min}, I_{IN} = -12mA$</td>
<td>-1.5</td>
<td></td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>High (82S16/116)$^4$</td>
<td>$V_{CC} = \text{Min}$</td>
<td>2.6</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Low$^6$</td>
<td>$I_{OL} = 16mA$</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>$I_{IH}$</td>
<td>High</td>
<td>$V_{CC} = \text{Max}$</td>
<td>1</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Low</td>
<td>$V_{IN} = 0.45V$</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>$I_{OLK}$</td>
<td>Leakage (82S17/117)$^6$</td>
<td>$V_{OUT} = 5.5V$</td>
<td>1</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{O(OFF)}$</td>
<td>Hi-Z state (82S16/116)$^8$</td>
<td>$V_{OUT} = 5.5V$</td>
<td>1</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{OS}$</td>
<td>Short-circuit (82S16/116)$^7$</td>
<td>$V_{CC} = \text{Max}, V_O = 0V$</td>
<td>-20</td>
<td>µA</td>
</tr>
<tr>
<td>$I_{CC}$</td>
<td>$V_{CC}$ supply current$^8$</td>
<td>$V_{CC} = \text{Max}$</td>
<td>80</td>
<td>mA</td>
</tr>
<tr>
<td>$C_{IN}$</td>
<td>Input</td>
<td>$V_{IN} = 5.0V$</td>
<td>5</td>
<td>pF</td>
</tr>
<tr>
<td>$C_{OUT}$</td>
<td>Output</td>
<td>$V_{OUT} = 2.0V$</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
AC ELECTRICAL CHARACTERISTICS

\[ R_1 = 2700 \Omega, \quad R_2 = 6000 \Omega, \quad C_L = 30 \text{pF} \]

N82S16/117, N82S16/17: \( 0^\circ \text{C} \leq T_A \leq +75^\circ \text{C}, \quad 4.75 \text{V} \leq V_{CC} \leq 5.25 \text{V} \)

S82S16/17: \( -55^\circ \text{C} \leq T_A \leq +125^\circ \text{C}, \quad 4.5 \text{V} \leq V_{CC} \leq 5.5 \text{V} \)

### AC ELECTRICAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TO</th>
<th>FROM</th>
<th>N82S16/17</th>
<th>N82S16/117</th>
<th>S82S16/17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min Typ1</td>
<td>Max</td>
<td>Min Typ1</td>
</tr>
<tr>
<td>( T_{AA} )</td>
<td>Access</td>
<td>Address</td>
<td>40</td>
<td>50</td>
<td>30</td>
</tr>
<tr>
<td>( T_{CE} )</td>
<td>Chip enable</td>
<td></td>
<td>30</td>
<td>40</td>
<td>15</td>
</tr>
<tr>
<td>( T_{CD} )</td>
<td>Disable</td>
<td>Output</td>
<td>30</td>
<td>40</td>
<td>15</td>
</tr>
<tr>
<td>( T_{WD} )</td>
<td>Valid time</td>
<td>Output</td>
<td>30</td>
<td>40</td>
<td>15</td>
</tr>
<tr>
<td>( T_{WSA} )</td>
<td>Setup and hold time</td>
<td>Output</td>
<td>20</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>( T_{WH} )</td>
<td>Hold time</td>
<td>Address</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T_{WS} )</td>
<td>Setup time</td>
<td>Write enable</td>
<td>40</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>( T_{WH} )</td>
<td>Hold time</td>
<td>Write enable</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T_{WS} )</td>
<td>Setup time</td>
<td>Write enable</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T_{WH} )</td>
<td>Hold time</td>
<td>Write enable</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T_{WP} )</td>
<td>Pulse width</td>
<td>Write enable9</td>
<td>30</td>
<td>15</td>
<td>25</td>
</tr>
</tbody>
</table>

### NOTES

1. All typical values are at \( V_{CC} = 5 \text{V}, \quad T_A = +25^\circ \text{C} \).
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 \( CE_1, CE_2 \) and \( CE_3 \).
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 \( CE_1, CE_2 \) and \( CE_3 \).
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

**LOADING CONDITION**

**VOLTAGE WAVEFORM**

**INPUT PULSES**

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

- **TCE**: Delay between beginning of Chip Enable low (with Address valid) and when Data Output becomes valid.
- **TCD**: Delay between when Chip Enable becomes high and Data Output is in off state.
- **TAA**: Delay between beginning of valid Address (with Chip Enable low) and when Data Output becomes valid.
- **TWSC**: Required delay between beginning of valid Chip Enable and beginning of Write Enable pulse.
- **TWHD**: Required delay between end of Write Enable pulse and end of valid Input Data.
- **TWP**: Width of Write Enable pulse.
- **TWSA**: Required delay between beginning of valid Address and beginning of Write Enable pulse.
- **TWSD**: Required delay between beginning of valid Data Input and end of Write Enable pulse.
- **TWD**: Delay between beginning of Write Enable pulse and when Data Output reflects complement of Data Input.
- **TWHC**: Required delay between end of Write Enable pulse and end of Chip Enable.
- **TWHA**: Required delay between end of Write Enable pulse and end of valid Address.

TIMING DIAGRAMS

**READ CYCLE**

- Address Access Time

**WRITE CYCLE**

- Chip Enable/Disable Times

Address Access Time

Chip Enable/Disable Times
DESCRIPTION

The Signetics 8X300 microprocessor is a high performance bipolar 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:

<table>
<thead>
<tr>
<th>DEVICE</th>
<th>MFG</th>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>U14</td>
<td>Signetics</td>
<td>N8X300I</td>
<td>Microprocessor</td>
<td>1</td>
</tr>
<tr>
<td>U2-U9</td>
<td>Signetics</td>
<td>N82S116B</td>
<td>RAM (256x1)</td>
<td>8</td>
</tr>
<tr>
<td>U15-16</td>
<td>Signetics</td>
<td>N82S1151</td>
<td>PROM (512x8)</td>
<td>2</td>
</tr>
<tr>
<td>U13, U21, U26, U27</td>
<td>Signetics</td>
<td>N8T32N</td>
<td>I/O Port (Tri-state)</td>
<td>4</td>
</tr>
<tr>
<td>U1</td>
<td>Signetics</td>
<td>N8T31N</td>
<td>Latch</td>
<td>1</td>
</tr>
<tr>
<td>U10-U11</td>
<td>Signetics</td>
<td>N8T26AB</td>
<td>Transceiver</td>
<td>2</td>
</tr>
<tr>
<td>U17-U20</td>
<td>Signetics</td>
<td>N74LS157B</td>
<td>Multiplexer</td>
<td>4</td>
</tr>
<tr>
<td>U23, U25</td>
<td>Signetics</td>
<td>N74LS74A</td>
<td>Flip-Flop</td>
<td>2</td>
</tr>
<tr>
<td>U12</td>
<td>Signetics</td>
<td>N74LS27A</td>
<td>NOR Gate</td>
<td>1</td>
</tr>
<tr>
<td>U22, U24</td>
<td>Signetics</td>
<td>N74LS00A</td>
<td>NAND Gate</td>
<td>2</td>
</tr>
<tr>
<td>Q1</td>
<td>Signetics</td>
<td>2N5320</td>
<td>Transistor</td>
<td>1</td>
</tr>
<tr>
<td>Q2</td>
<td>Dale</td>
<td>2N2222</td>
<td>Transistor</td>
<td>1</td>
</tr>
<tr>
<td>RN1-RN3</td>
<td>Dale</td>
<td>CSP-10E-01-102-K</td>
<td>Resistor network</td>
<td>3</td>
</tr>
<tr>
<td>R1, R3, R4</td>
<td>Dale</td>
<td>1k Ω 1/4W</td>
<td>Resistor</td>
<td>3</td>
</tr>
<tr>
<td>R5</td>
<td>Dale</td>
<td>4k Ω 1/4W</td>
<td>Resistor</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>Dale</td>
<td>8k Ω 1/4W</td>
<td>Resistor</td>
<td>1</td>
</tr>
<tr>
<td>C2-C20, C22-C30</td>
<td>Dale</td>
<td>0.1μF</td>
<td>Capacitor</td>
<td>29</td>
</tr>
<tr>
<td>C1, C21</td>
<td>Dale</td>
<td>22μF</td>
<td>Capacitor</td>
<td>2</td>
</tr>
<tr>
<td>Z1</td>
<td>Dale</td>
<td>8.00 MHz</td>
<td>Crystal</td>
<td>1</td>
</tr>
<tr>
<td>J104</td>
<td>Berg</td>
<td>65616-134</td>
<td>Connector</td>
<td>1</td>
</tr>
<tr>
<td>J101</td>
<td>Berg</td>
<td>65616-150</td>
<td>Connector</td>
<td>1</td>
</tr>
<tr>
<td>J105</td>
<td>Berg</td>
<td>65616-120</td>
<td>Connector</td>
<td>4</td>
</tr>
<tr>
<td>Robinson-Nugent</td>
<td>ICN-163-53</td>
<td>ICN-246-54</td>
<td>Socket</td>
<td>2</td>
</tr>
<tr>
<td>Robinson-Nugent</td>
<td>ICN-163-53</td>
<td>ICN-246-54</td>
<td>Socket</td>
<td>7</td>
</tr>
<tr>
<td>Robinson-Nugent</td>
<td>ICN-163-53</td>
<td>ICN-246-54</td>
<td>Strip socket</td>
<td>2</td>
</tr>
<tr>
<td>S1, S2</td>
<td>Amp</td>
<td>435640-5</td>
<td>Dip switch</td>
<td>2</td>
</tr>
<tr>
<td>S5, S6</td>
<td>Alco</td>
<td>MSTS-104D</td>
<td>Toggle switch</td>
<td>2</td>
</tr>
<tr>
<td>S3, S4</td>
<td>Digitast</td>
<td>ST</td>
<td>Momentary switch</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>Smith</td>
<td>230 (red)</td>
<td>Binding post</td>
<td>1</td>
</tr>
<tr>
<td>Smith</td>
<td>Smith</td>
<td>230 (black)</td>
<td>Binding post</td>
<td>1</td>
</tr>
<tr>
<td>SAE</td>
<td>SAE</td>
<td>CPH 8100-100</td>
<td>50/100 edge connector</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PC board</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Packing case</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Assembly manual</td>
<td>1</td>
</tr>
</tbody>
</table>

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.

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 BOC (J105 pin 18) be tied low for most tests and that 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 BIC overrides 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 combinations 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. SS 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).
To begin the test sequence, the DIP switches are loaded with 700000 (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 if 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
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>INSTRUCTION</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>6 06374</td>
<td>XMIT -4, R6</td>
<td>Initialize Counter</td>
</tr>
<tr>
<td>001</td>
<td>6 11000</td>
<td>XMIT 0, R11</td>
<td>Initialize Port Register</td>
</tr>
<tr>
<td>002</td>
<td>6 00001</td>
<td>XMIT 1, AUX</td>
<td>Load Increment</td>
</tr>
<tr>
<td>003</td>
<td>1 06006</td>
<td>Q1 ADD R6, R6</td>
<td>Incr Count</td>
</tr>
<tr>
<td>004</td>
<td>1 11011</td>
<td>ADD R11, R11</td>
<td>Incr Port Number</td>
</tr>
<tr>
<td>005</td>
<td>0 11007</td>
<td>MOVE R11, IVL</td>
<td>Select Port</td>
</tr>
<tr>
<td>006</td>
<td>0 27001</td>
<td>MOVE LB, R1</td>
<td>Read Port</td>
</tr>
<tr>
<td>007</td>
<td>6 27000</td>
<td>XMIT 0, LB</td>
<td>Clear Port</td>
</tr>
<tr>
<td>010</td>
<td>0 01027</td>
<td>MOVE R1, LB</td>
<td>Write Port</td>
</tr>
<tr>
<td>011</td>
<td>5 06003</td>
<td>NZT R6, Q1</td>
<td>All Done?</td>
</tr>
<tr>
<td>012</td>
<td>6 07001</td>
<td>XMIT 1, IVL</td>
<td>Address Port 1</td>
</tr>
<tr>
<td>013</td>
<td>0 27001</td>
<td>MOVE LB, R1</td>
<td>Read RAM Addr</td>
</tr>
<tr>
<td>014</td>
<td>6 07002</td>
<td>XMIT 2, IVL</td>
<td>Addr Port 2</td>
</tr>
<tr>
<td>015</td>
<td>0 27002</td>
<td>MOVE LB, R2</td>
<td>Read RAM Data</td>
</tr>
<tr>
<td>016</td>
<td>0 01017</td>
<td>MOVE R1, IVR</td>
<td>Addr RAM</td>
</tr>
<tr>
<td>017</td>
<td>0 02037</td>
<td>MOVE R2, RB</td>
<td>Write RAM</td>
</tr>
<tr>
<td>020</td>
<td>6 02000</td>
<td>XMIT 0, R2</td>
<td>Clear R2</td>
</tr>
<tr>
<td>021</td>
<td>6 07001</td>
<td>XMIT 1, IVL</td>
<td>Addr Port 1</td>
</tr>
<tr>
<td>022</td>
<td>0 37002</td>
<td>MOVE RB, R2</td>
<td>Read RAM</td>
</tr>
<tr>
<td>023</td>
<td>0 02027</td>
<td>MOVE R2, LB</td>
<td>Output Data to Port 1</td>
</tr>
</tbody>
</table>

**BEGINNING OF MEMORY TEST**

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>INSTRUCTION</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>024</td>
<td>6 01000</td>
<td>XMIT 0, R1</td>
<td>Initialize RAM Addr</td>
</tr>
<tr>
<td>025</td>
<td>6 07001</td>
<td>Q2 XMIT 1, IVL</td>
<td>Addr Port 1</td>
</tr>
<tr>
<td>026</td>
<td>0 01027</td>
<td>MOVE R1, LB</td>
<td>Output RAM Addr</td>
</tr>
<tr>
<td>027</td>
<td>0 27017</td>
<td>MOVE LB, IVR</td>
<td>Addr RAM</td>
</tr>
<tr>
<td>030</td>
<td>6 02000</td>
<td>XMIT 0, R2</td>
<td>Initialize RAM Data</td>
</tr>
<tr>
<td>031</td>
<td>6 07002</td>
<td>XMIT 2, IVL</td>
<td>Addr Port 2</td>
</tr>
<tr>
<td>032</td>
<td>0 02027</td>
<td>Q3 MOVE R2, LB</td>
<td>Output RAM Data</td>
</tr>
<tr>
<td>033</td>
<td>0 02037</td>
<td>MOVE R2, RB</td>
<td>Write to RAM</td>
</tr>
<tr>
<td>034</td>
<td>0 02000</td>
<td>MOVE R2, AUX</td>
<td>Move Data to AUX</td>
</tr>
<tr>
<td>035</td>
<td>3 37000</td>
<td>XOR RB, AUX</td>
<td>Compare</td>
</tr>
<tr>
<td>036</td>
<td>5 00032</td>
<td>NZT AUX, Q3</td>
<td>Test</td>
</tr>
<tr>
<td>037</td>
<td>6 11000</td>
<td>XMIT 0, R11</td>
<td>Set up Return Indicator</td>
</tr>
<tr>
<td>040</td>
<td>7 00050</td>
<td>JMP DELAY</td>
<td>Delay for display purposes</td>
</tr>
<tr>
<td>041</td>
<td>6 00001</td>
<td>XMIT 1, AUX</td>
<td>Load Incr</td>
</tr>
<tr>
<td>042</td>
<td>1 02002</td>
<td>ADD R2, R2</td>
<td>Incr Data</td>
</tr>
<tr>
<td>043</td>
<td>5 02032</td>
<td>NZT R2, Q3</td>
<td>Done?</td>
</tr>
<tr>
<td>044</td>
<td>1 01001</td>
<td>ADD R1, R1</td>
<td>Incr Addr</td>
</tr>
<tr>
<td>045</td>
<td>6 11001</td>
<td>XMIT 1, R11</td>
<td>Set up Return Indicator</td>
</tr>
<tr>
<td>046</td>
<td>7 00050</td>
<td>JMP DELAY</td>
<td>Delay</td>
</tr>
<tr>
<td>047</td>
<td>7 00025</td>
<td>JMP Q2</td>
<td>Repeat</td>
</tr>
<tr>
<td>050</td>
<td>6 03000</td>
<td>DELAY XMIT 0, R3</td>
<td></td>
</tr>
<tr>
<td>051</td>
<td>6 04000</td>
<td>XMIT 0, R4</td>
<td></td>
</tr>
<tr>
<td>052</td>
<td>6 00001</td>
<td>XMIT 1, Aux</td>
<td></td>
</tr>
<tr>
<td>053</td>
<td>1 03003</td>
<td>D1 ADD R3, R3</td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>1 04004</td>
<td>D2 ADD R4, R4</td>
<td></td>
</tr>
<tr>
<td>055</td>
<td>5 04054</td>
<td>NZT R4, D2</td>
<td></td>
</tr>
<tr>
<td>056</td>
<td>5 03053</td>
<td>NZT R3, D1</td>
<td></td>
</tr>
<tr>
<td>057</td>
<td>4 11060</td>
<td>RTN XEC *+1, R11</td>
<td></td>
</tr>
<tr>
<td>060</td>
<td>7 00041</td>
<td>JMP 041</td>
<td></td>
</tr>
<tr>
<td>061</td>
<td>7 00047</td>
<td>JMP 047</td>
<td></td>
</tr>
</tbody>
</table>

Table 3  SYSTEM TEST
SALES OFFICES
## DISTRIBUTORS

### ALABAMA
- **Huntsville**
  - Hamilton/Avnet Electronics
  - Phone: (205) 931-1107

### ARIZONA
- **Phoenix**
  - Philips Electronics (Industrial Products)
  - Phone: (602) 443-4411
- **Phoenix**
  - Hamilton/Avnet Electronics
  - Phone: (602) 728-7813
  - Liberty Electronics
  - Phone: (602) 257-1792

### CALIFORNIA
- **Carlsbad**
  - Metalectronics Semiconductors
  - Phone: (619) 481-0675
- **San Diego**
  - Hamilton/Avnet Electronics
  - Phone: (714) 258-9908
  - Calderon City
  - Hamilton/Avnet Electronics
  - Phone: (213) 518-2103

### COLORADO
- **Denver**
  - Philips Electronics
  - Phone: (303) 812-8233

### FLORIDA
- **Tampa**
  - Airline Products
  - Phone: (813) 292-5161

### KANSAS
- **Kansas City**
  - Hamilton/Avnet Electronics
  - Phone: (816) 504-7010
  - Buffalo
  - Hamilton/Avnet Electronics
  - Phone: (716) 275-7851

### MARYLAND
- **Rockville**
  - Philips Electronics
  - Phone: (301) 797-5800

### MASSACHUSETTS
- **Boston**
  - Philips Electronics
  - Phone: (617) 854-3011

### MICHIGAN
- **Detroit**
  - Philips Electronics
  - Phone: (313) 201-0300

### MISSOURI
- **St. Louis**
  - Philips Electronics
  - Phone: (314) 226-1700

### NEW JERSEY
- **Newark**
  - Philips Electronics
  - Phone: (973) 286-6411

### NEW MEXICO
- **Albuquerque**
  - The Skyline Company, Inc.
  - Phone: (505) 925-6000

### NEW YORK
- **New York City**
  - Philips Electronics
  - Phone: (212) 272-2187

### NORTH CAROLINA
- **Durham**
  - Philips Electronics
  - Phone: (919) 941-3801

### OHIO
- **Cleveland**
  - Philips Electronics
  - Phone: (216) 841-3116

### REPRESENTATIVES

### ALABAMA
- **Huntsville**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### ARIZONA
- **Phoenix**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### CALIFORNIA
- **Fresno**
  - Philips Electronics
  - Phone: (559) 424-5687

### COLORADO
- **Denver**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### FLORIDA
- **Tampa**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### KANSAS
- **Kansas City**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### MARYLAND
- **Silver Spring**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### MASSACHUSETTS
- **Waltham**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### MICHIGAN
- **Dearborn**
  - Sherman Oaks Marketing
  - Phone: (213) 250-2547

### MISSOURI
- **St. Louis**
  - Sherman Oaks Marketing
  - Phone: (314) 731-3144

### NEW JERSEY
- **Newark**
  - Sherman Oaks Marketing
  - Phone: (973) 286-6411

### NEW MEXICO
- **Albuquerque**
  - Sherman Oaks Marketing
  - Phone: (505) 925-6000

### NEW YORK
- **New York City**
  - Sherman Oaks Marketing
  - Phone: (212) 469-5687

### NORTH CAROLINA
- **Durham**
  - Sherman Oaks Marketing
  - Phone: (919) 941-3801

### OHIO
- **Cleveland**
  - Sherman Oaks Marketing
  - Phone: (216) 841-3116

### SIGNETICS PRODUCTS

## WORLDWIDE

### ARGENTINA
- **Buenos Aires**
  - Philips Industrial Products
  - Phone: (54) 226-7847

### AUSTRIA
- **Vienna**
  - Philips Industrial Products
  - Phone: (43) 476-3653

### BELGIUM
- **Brussels**
  - Philips Industrial Products
  - Phone: (32) 295-8200

### CANADA
- **Montreal**
  - Philips Industrial Products
  - Phone: (514) 546-8300

### FRANCE
- **Paris**
  - Philips Industrial Products
  - Phone: (01) 438-72-02

### GERMANY
- **Berlin**
  - Philips Industrial Products
  - Phone: (030) 388-9001

### ITALY
- **Milan**
  - Philips Industrial Products
  - Phone: (02) 1118-22

### SWITZERLAND
- **Zurich**
  - Philips Industrial Products
  - Phone: (01) 438-72-02

### JAPAN
- **Tokyo**
  - Philips Industrial Products
  - Phone: (03) 295-8300

### SOUTH KOREA
- **Seoul**
  - Philips Industrial Products
  - Phone: (02) 438-3653

### THAILAND
- **Bangkok**
  - Philips Industrial Products
  - Phone: (02) 438-3653

### SPAIN
- **Barcelona**
  - Philips Industrial Products
  - Phone: (03) 295-8300

### SWEDEN
- **Stockholm**
  - Philips Industrial Products
  - Phone: (08) 2585-978

### UNITED KINGDOM
- **London**
  - Philips Industrial Products
  - Phone: (01) 4422-11

### UNITED STATES
- **Arkansas**
  - Philips Industrial Products
  - Phone: (501) 438-3653

### VENEZUELA
- **Caracas**
  - Philips Industrial Products
  - Phone: (02) 438-3653