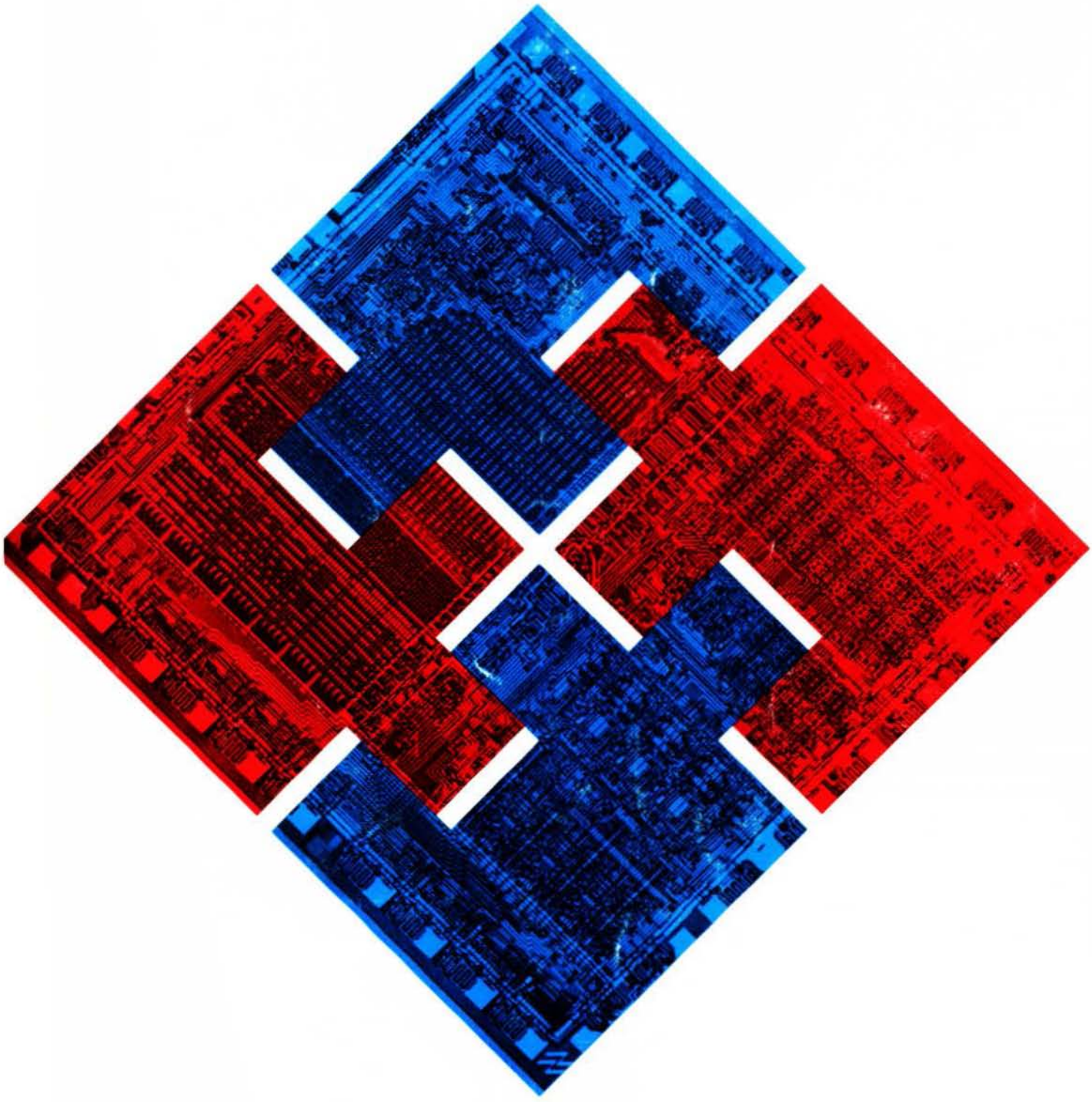
SC/MP Microprocessor Applications Handbook



National Semiconductor

Publication Number 420305239-001A

SC/MP Applications Handbook

PREFACE

In conjunction with other support documents (listed below), this Applications Guide provides the user with sufficient information to build, checkout, and utilize a wide variety of SC/MP-based systems. The information is organized in capsule form; thus, the designer can, with minimum effort, expand, modify, or customize a given application. The applications (chapter 2) are organized by class – Analog-to-Digital/Digital-to-Analog Systems, Keyboard/Display Systems, Multiprocessor Systems, and so on. Chapter 1 and the appendixes provide general design data as regards the instruction set, addressing modes, input/output capabilities, interrupt structures, and other applications-related features.

To complete your SC/MP support library, the following documents are presently available:

- Data Sheet, ISP-8A/500D Single-Chip 8-bit MicroProcessor (SC/MP) – publication number 420305227-001. *Provides electrical characteristics and functional overview of SC/MP chip.*
- SC/MP Technical Description – publication number 4200079. *Comprises comprehensive descriptions of functional details, general interfacing characterisitcs, supporting hardware, and systems information.*
- SC/MP Assembly Language Programming Manual – order number ISP-8S/994Y.*Contains comprehensive overview of the SC/MP Microprocessor as it relates to assembly language programming and detailed descriptions of the assembly language, sundry source statements, programming techniques, and assembly input/output formats.*

The information in this handbook is believed to be reliable; nevertheless, the National Semiconductor Corporation does not assume responsibility for inaccuracies and the material presented is subject to change without notice. Furthermore, such information does not convey to the user of semiconductor devices described any license under the patent rights of the National Semiconductor Corporation or others.

February 1977R

2900 Semiconductor Drive
Santa Clara, California 95051

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

Chapter 1

SC/MP AS A GENERAL PURPOSE APPLICATIONS MICROPROCESSOR

INTRODUCTION

This document defines the internal architecture, the pinouts, and the interfacing techniques of the SC/MP microprocessor from an applications point-of-view; it is also addressed to the concepts, principles, hook-up detail, and general implementation procedures that relate to the many applications for which SC/MP is suitable. It is assumed that the user of this document is somewhat familiar with SC/MP hardware and software. If additional information is required in either of these areas, the following support documents are available:

- Data Sheet, ISP-8A/500D Single-Chip 8-bit Microprocessor (SC/MP) – publication number 420305227-001. *Provides electrical characteristics and functional overview of SC/MP chip.*
- SC/MP Technical Description – publication number 4200079. *Comprises comprehensive descriptions of functional details, general interfacing characteristics, supporting hardware, and systems information.*
- SC/MP Assembly Language Programming Manual– order number ISP-8S/994Y. *Contains comprehensive overview of the SC/MP Microprocessor as it relates to assembly language programming and detailed descriptions of the assembly language, sundry source statements, programming techniques, and assembly input/output formats.*

APPLICATIONS OVERVIEW OF SC/MP

SC/MP is a single-chip 8-bit microprocessor with a 16-bit addressing capability that provides the user with a simple, cost-effective, and general-purpose tool for implementing a wide range of applications. The basic functions of SC/MP are shown in figure 1-1. The following sections are addressed to the purpose and use of these basic functions as they relate to the broad spectrum of SC/MP applications.

SC/MP Timing

For applications where timing is not particularly critical, only a capacitor is required for the clock – all other timing circuits are an integral part of the chip. If the application requires precision clock control, the capacitor can be replaced with a crystal – shown connected by broken lines in figure 1-1. In either case, no external timing circuits (clock drivers, phase splitters, buffers, and so forth) are required. Refer to appendix A for information on using a capacitor or a crystal, and for external clock considerations.

Sense Lines

SC/MP uses two sense lines (Sense A and Sense B) as its "eyes" to see what is happening external to the processor; the sense inputs shown in figure 1-1 are an integral part of the chip. Under program control, either or both sense lines

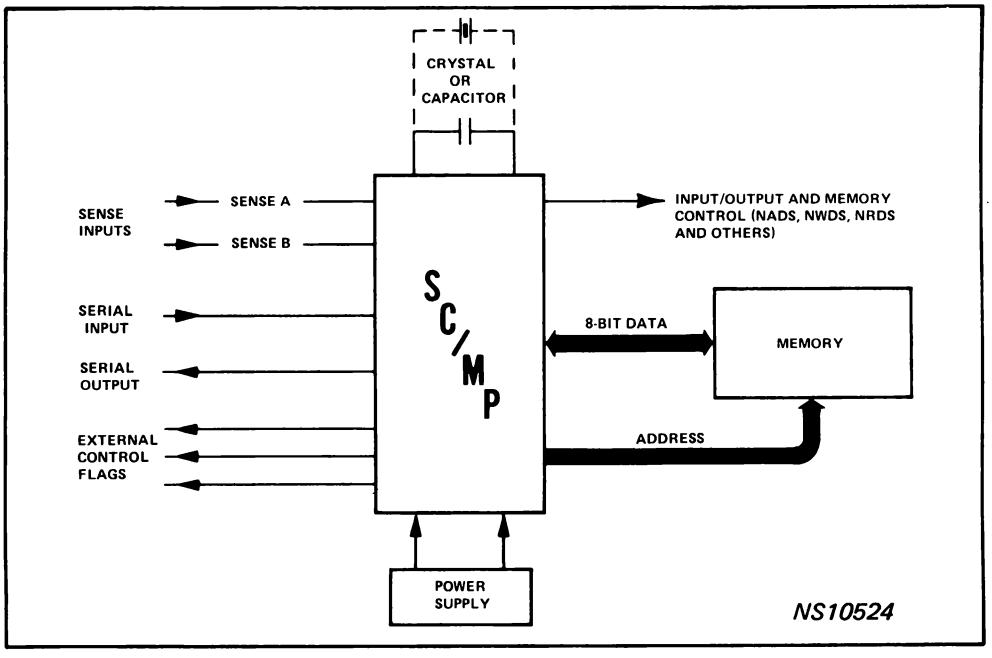


Figure 1–1. Basic Functions of SC/MP

can be examined and a decision based on the logic state of one or both inputs can be made.

The Sense A line serves a dual function in that it is used as the interrupt input; when Sense A is used as an interrupt, SC/MP can monitor asynchronous events while running other programs. (Refer to appendix C for implementation detail of SC/MP interrupts.) In certain applications, the Sense A line can also be used for polling.

Serial Input/Output

The serial interface capability of SC/MP is provided by a single 8-bit register (the Extension Register) and a "ninth-bit" output latch for the least significant digit of the word. The serial input/output capability is especially useful for Teletype® applications, since the Teletype data are in 8-bit format – a parity bit plus a 7-bit code. The serial interface is also cost-effective in inexpensive analog-to-digital systems, X-Y plotters, display systems, data-acquisition systems, and a host of other applications where inputs are generated in serial order and the output device can be driven serially.

Control Flags

Discrete control of events that occur external to the microprocessor are provided by three control flags (0, 1, and 2); these TTL-compatible outputs are built into the SC/MP chip. Under software control, each flag output can produce pulses or fixed voltage levels; thus, lamps, motors, relays, and other similar devices can be serviced as required by the application. Since any combination of the flags can be set with a simple series of instructions, there is a significant savings in external decoding logic.

8-Bit Data Bus

The 8-bit data bus provides a means of bidirectional data transfer between the Accumulator, which is internal to SC/MP, and standard memory components or peripheral devices, which are external to SC/MP. Valid input data are expected to be on the bus at negative read-strobe (NRDS) time, whereas output data are valid on the bus at negative write-strobe (NWDS) time. The four most significant address bits (AD 12, AD 13, AD 14, and AD 15) and four input/output status signals (RFLG, IFLG, DFLG, and HFLG) are multiplexed on the data bus and appear at negative address-strobe (NADS) time; if the application requires, the address and status signals can be externally latched. (Note: In applications that use Direct Memory Access (DMA), data transfers between memory and a peripheral can be made without entering the data port of SC/MP; that is, SC/MP does not actively participate in the actual transfer of data. Refer to the SC/MP Technical Description for details of DMA operation.)

® Registered trademark of the Teletype Corporation.

12-Bit Address Bus

SC/MP provides 12 dedicated address lines (AD 00 through AD 11) that are internally latched – an applications feature that can save much external logic. As previously indicated, the 4 most significant address bits (AD 12 through AD 15) are time-multiplexed on the 8-bit data bus; thus, a SC/MP system can be expanded easily to 65,536 bytes of memory.

Power Requirements

For direct interface with TTL and MOS devices, both +5-volt and -7-volt sources are required. From an applications viewpoint, it is advantageous to use low-power TTL and MOS devices, since the SC/MP interface can be implemented without buffering.

Putting the Basics of SC/MP to Use

Before looking at the internal architecture of SC/MP, let us first see how an applications concept can be developed by using nothing more than the basic features just described and some simple software. Consider a "fire alarm system" with a fire-sensing device connected to the Sense A/interrupt line. The interrupt capability of the Sense A line is enabled by software and, on each instruction fetch to memory, the interrupt input is interrogated. As long as there is no fire, the interrupt status does not change and SC/MP stays in the main program. If a fire occurs, the interrupt input becomes active (high), and on the next instruction fetch, SC/MP is directed to another program; this program is written by the user and is called an *interrupt service routine*. The interrupt service routine can be written to serve any or all of several functions – turning on the fire alarms, playback of prerecorded evacuation procedures and perhaps opening emergency exits, turning on the sprinkler system, calling the fire department, and so on. All of these functions can be performed by software control of flags 0, 1, and 2. For instance, the alarm system and playback of evacuation procedures might be activated simply by setting flag 0 from a logic '0' to a logic '1'. A series of pulses on flag 1 might be used to open one or more emergency exits and to turn on the sprinkler systems. Flag 2 might be used to activate an automatic phone dialer – a coded pulse train could indicate the number to be dialed. The phone systems could also be activated and the dial-up code transmitted via the serial input/output port of SC/MP. Except for memory and external interface circuits, all hardware required to implement the fire control system is resident in the SC/MP chip; thus, optimal cost-effectiveness is obtained.

SC/MP Hardware

In the preceding section, SC/MP is defined in basic terms with little emphasis on the internal architecture of SC/MP. To view SC/MP from a general-purpose applications perspective, a study of the CPU architecture and a description of the pinouts are in order. Figure 1-2 shows the functional subdivisions and identifies each input/output pin of the SC/MP chip. In subsequent sections, all functions and pinouts that are related directly to applications are described in terms of both hardware and software.

Accumulator Register

Almost any data movement within the SC/MP system involves the 8-bit Accumulator; thus, the Accumulator is used in every type of application. In terms of interface and software control, functions of the Accumulator can be summarized as follows.

- Data transfers to and from memory are made via the Accumulator.
- Under software control, the low-order or high-order data byte of any Pointer Register can be exchanged directly with the contents of the Accumulator; data can also be exchanged between the Extension Register and the Accumulator.
- Data can be copied from the Status Register to the Accumulator and vice versa.
- Results of all operations performed by the Arithmetic Logic Unit are left in the Accumulator.

Pointer Registers

Except for Pointer Register 0 (P0), which is dedicated for use as the Program Counter (PC), the remaining three Pointer Registers (P1, P2, and P3) are available for addressing memory and other peripheral devices. In a given application, a Pointer Register might be used as follows: 16 bits (2 bytes) of address data are transferred via the Accumulator to a designated Pointer Register – say, P1. Now, when an 'LD 1' Instruction is executed, SC/MP is instructed to load 8-bits of data from the external device whose address is specified by the contents of P1. Another pointer – P3, for example – could be loaded with the memory location in which the data are to be stored, that is, an 'ST 3' Instruction. When this instruction is executed, the foregoing data are transferred from the Accumulator to the memory location specified by P3.

From a programming point-of-view, certain conventions regarding pointer register assignments are beneficial; some typical assignments are as follows:

Pointer Register 1 - - - - - - - - - ROM Pointer
Pointer Register 2 - - - - - - - - - RAM Pointer
Pointer Register 3 - - - - - - - - - Subroutine Pointer

As previously indicated, the low-order or the high-order byte of any Pointer Register can be exchanged with the Accumulator; also, the 16-bit content of the Program Counter (PC) can be exchanged with any of the other Pointer Registers. Not only are these exchanges useful in implementing the addressing schemes of SC/MP, the 16-bit exchange is particularly useful in executing subroutine calls. For example, using the foregoing convention, Pointer Register 3 is loaded with the memory location that just precedes the subroutine address; that is, if the subroutine address is X'FE15, then, Pointer Register 3 is set to X'FE14. The content of P3 now is exchanged with the Program Counter, and since the PC is incremented before the instruction fetch, the instruction at address X'FE15 is executed. During subroutine nesting, it is important to save the contents of P3 because they are the only return link to the basic program. This is easily accomplished via the Accumulator and two store instructions – one for the low-order byte and one for the high-order byte. The following examples assume that SUBR is the label on the first instruction of the subroutine.

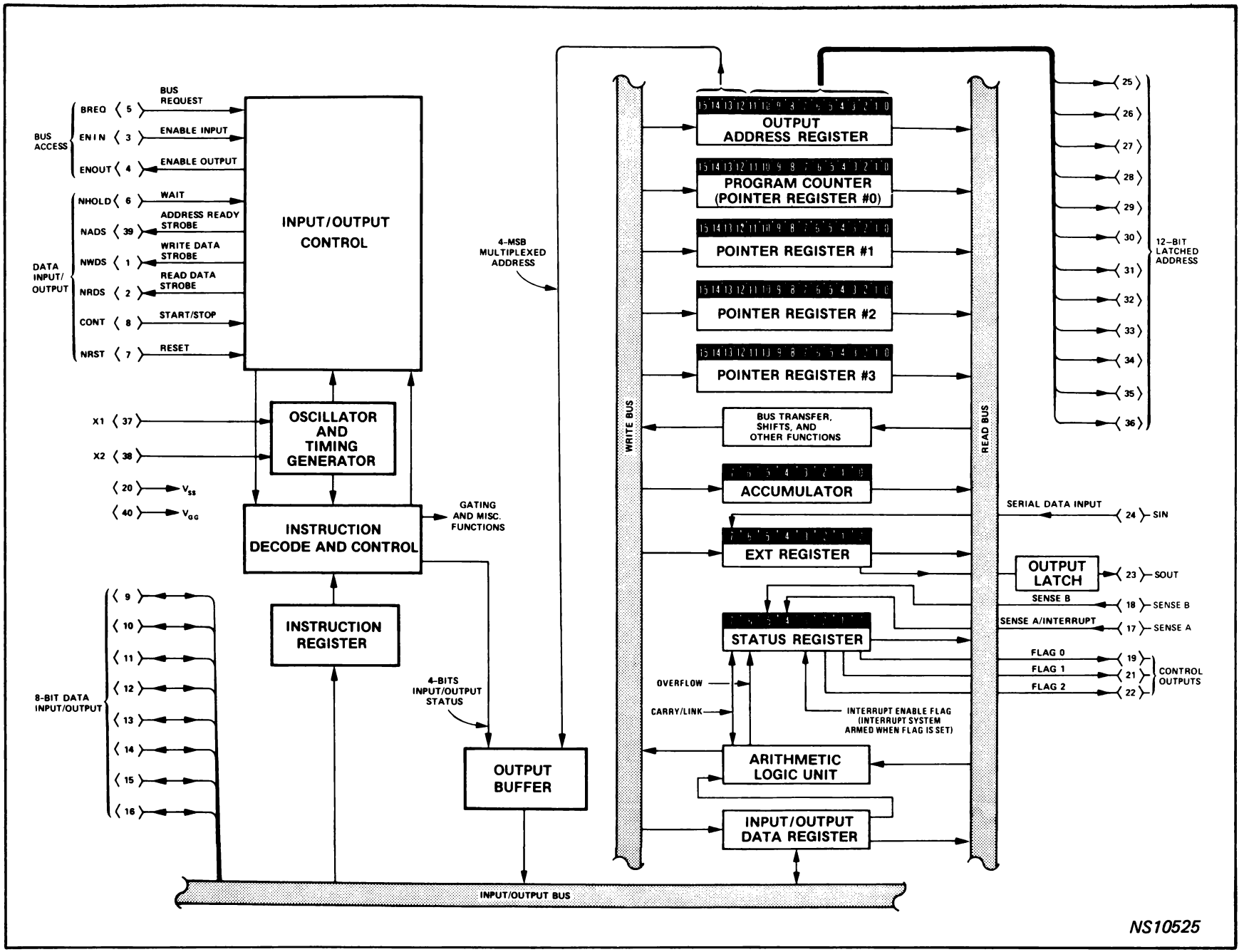


Figure 1–2. CPU Architecture and Pinouts of SC/MP

SUBROUTINE JUMP

```
        FE14    SUBR 1    =       SUBR-1
        C414              LDI     L(SUBR 1)     ;LOAD LOWER SUBROUTINE ADDRESS
        33                XPAL    P3            ;TRANSFER LOWER TO P3L
        C4FE              LDI     H(SUBR 1)     ;LOAD UPPER SUBROUTINE ADDRESS
        37                XPAH    P3            ;TRANSFER UPPER TO P3H
        3F                XPPC    P3            ;EXCHANGE PC AND P3
SUBROUTINE RETURN
        3F                XPPC    P3            ;RETURN FROM SUBROUTINE EXCHANGE
```

If multilevel subroutines are used, the current contents of the Pointer Register should be saved on the top of the stack and should be restored upon return from the subroutine.

```
        C414              LDI     L(SUBR 1)     ;LOAD LOWER SUBROUTINE ADDRESS
        33                XPAL    P3            ;TRANSFER TO P3
        CEFF              ST      @-1(P2)       ;SAVE P3L ON STACK
        C4FE              LDI     H(SUBR 1)     ;LOAD UPPER SUBROUTINE ADDRESS
        37                XPAH    P3            ;TRANSFER TO P3H
        CEFF              ST      @-1(P2)       ;SAVE P3H ON STACK
        3F                XPPC    P3            ;JUMP TO SUBROUTINE
        C6FF              LD      @1(P2)        ;RETURN FROM SUBROUTINE, LOAD
                                                ;P3H FROM STACK
        37                XPAH    P3            ;TRANSFER TO P3H
        C601              LD      @1(P2)        ;LOAD P3L FROM STACK
        33                XPAL    P3            ;TRANSFER P3L FOR RETURN TO
                                                ;CALLING SUBROUTINE
```

A subroutine call can also be implemented via the Jump to Subroutine Instruction (JS). The SC/MP assembler treats this pseudo instruction the same as any other assembly-language statement except that, in this case, one or more machine-language instructions are generated. Following is an example:

JUMP TO SUBROUTINE (JS)

	Operation	Operands
Format:	JS	ptr, expression
Memory:	7 bytes	
Generated Code:	LDI	H(expression)
	XPAH	ptr
	LDI	L(expression)–1
	XPAL	ptr
	XPPC	ptr

When a Jump to Subroutine is invoked, the code generated results in the setting of the specified Pointer Register (ptr) to the value, expression-1. In calculating this value, the memory page structure of SC/MP must be considered—refer to SC/MP Assembly Language Programming Manual. The contents of the Pointer Register are exchanged with the To implement multilevel (nested) subroutines, a memory "stack" is required. In accordance with the foregoing assignments, Pointer Register 2 can be used as the stack pointer in RAM. The address loaded into P2 points to the top of the stack; this address is a location in read/write memory. contents of the Program Counter. As a result, the next instruction to be executed will be at the location addressed by the expression. The Pointer Register will contain the address of the XPPC Instruction, allowing a subroutine return to the instruction that follows.

If P3 is used as a subroutine pointer and its contents are not disturbed, the subroutine can be called repeatedly without reloading P3. The following setup shows how this can be accomplished.

```
        SIN:
          •     •
          •     •
          •     •
          •     •
        3F        XPPC    P3    ;SUBROUTINE RETURN
        90FD      JMP     SIN   ;FOR REENTRY
```

Extension Register

Basically, the 8-bit Extension Register is a backup for the Accumulator as the Accumulator can be loaded from the Extension Register with a Load AC from Extension Instruction (LDE) or the contents of the two registers can be exchanged with an Exchange AC and Extension Instruction (XAE). Also a serial input/output can be implemented via the Extension Register and the Serial Input/Output Instruction (SIO). Technically, the Extension Register can be thought of as a 9-bit register; the additional bit in the output latch is useful for implementing 'starts' and 'stops' while still maintaining 8 bits of data in the register. A simple sequence of instructions to initialize the output latch is shown below.

```
INITIALIZE:   LI     0      ;LOAD ACCUMULATOR WITH ALL ZEROES
              XAE             ;TRANSFER CONTENTS OF ACCUMULATOR TO
                              ;E-REGISTER
              SIO             ;SET OUTPUT LATCH TO ZERO
```

In the first Load Immediate Instruction (LI), the Accumulator is loaded with all zeroes; the second Exchange AC and Extension Instruction (XAE) fills the 8-bit Extension Register with zeroes. Now, a Serial Input/Output Instruction (SIO) is executed; this right-shifts the Extension Register one position and brings the data at the serial input (SIN) port into the most significant bit position, while shifting a '0' into the output latch. (Note: The Extension Register also can be used for addressing – refer to "Addressing Capabilities and Instruction Set of SC/MP," located later in this section, for additional information about addressing.)

Status Register

Conditional responses of SC/MP are based on the logic '0' or logic '1' states of the 8 bits in this register. As shown in figure 1-2, bits 0,1, and 2, respectively, are dedicated to control flags 0, 1, and 2. One or more of the control flags are set by loading a bit pattern into the Accumulator and then copying the contents of the Accumulator to the Status Register; refer to series of instructions that follows.

```
1     PFLG:   LDI   01    ;SET LSB OF ACCUMULATOR TO '1'
2             CAS         ;SET FLAG 0 to '1'
3             LDI   00    ;SET CONTENTS OF ACCUMULATOR
                          ;EQUAL TO 0
4             CAS         ;RESET FLAG 0 FROM LOGIC '1' TO
                          ;LOGIC '0'
•
•
•
21    SFLG:   LDI   03    ;SET BIT POSITIONS '0' AND '1' OF
                          ;ACCUMULATOR TO LOGIC '1'
22            CAS         ;SET FLAGS 0 AND 1 SIMULTANEOUSLY
```

From the preceding instructions, it is readily seen that flags 0, 1, and 2 are software-controlled to produce a series of pulses or a DC level at the output pins in whatever timing sequence the application requires. The state of the flags can be tested by copying the contents of the Status Register to the Accumulator and by using a masking or bit-testing operation – similar to the Sense A/Sense B program that follows.

```
1        CSA              ;COPY CONTENTS OF STATUS REGISTER INTO ACCUMULATOR
2        ANI     010      ;MASK TO TEST SENSE A (BIT 4)
3        JNZ     SASET    ;JUMP TO SASET IF SENSE A = 1
•                         ;CONTINUE PROGRAM IF SENSE A = 0
•
•
11       CSA              ;COPY CONTENTS OF STATUS REGISTER INTO ACCUMULATOR
12       ANI     020      ;MASK TO TEST SENSE B (BIT 5)
13       JNZ     SBSET    ;JUMP TO SBSET IF SENSE B = 1
•                         ;CONTINUE PROGRAM IF SENSE B = 0
•
•
21       CSA              ;COPY CONTENTS OF STATUS REGISTER INTO ACCUMULATOR
22       ANI     030      ;MASK TO TEST SENSE A AND SENSE B (BITS 4 AND 5)
23       JNZ     ORSET    ;JUMP TO ORSET IF SENSE A OR SENSE B IS EQUAL TO '1'
                          ;CONTINUE PROGRAM IF BOTH SENSE A AND SENSE B
                          ;EQUALS '0'
```

As shown in the foregoing code, the operational sequence for testing each sense input is the same – the only difference being that for Sense A the mask coincides with bit 4 and that for Sense B it coincides with bit 5. When the two sense lines are tested simultaneously (lines 21, 22, and 23), the program does not immediately identify which sense line is high but simply jumps to a location (ORSET) if either Sense A or Sense B is set. This method of testing might be useful in a polling system where time is critical and it takes too long to check discretely each sense line every time the program circulates through the polling loop. By simultaneously checking both inputs and by jumping to an appropriate subroutine, software can be used to discriminate between Sense A and Sense B – if, in fact, one of the sense inputs is high.

Many applications require an interrupt input, and the Sense A line can be used for this purpose. The SC/MP interrupt system is enabled simply by using an Enable Interrupt Instruction (IEN) to set bit 3 in the Status Register; alternately, the interrupt system can be enabled, first, by setting bit 3 of the Accumulator to a logic '1' and, then, by executing a copy AC to Status Instruction (CAS). In either case, with bit 3 of the status register set to a logic '1', the Sense A line becomes an interrupt input – refer to appendix C for implementation detail of the interrupts. The interrupt system can be disabled in either of two ways: (1) a Disable Interrupts Instruction (DINT) can be used to set bit 3 of the Status Register to a logic '0', or (2) the interrupt system can be disabled, first, by setting bit 3 of the Accumulator to '0' and, then, by executing a Copy AC to Status Instruction (CAS). With bit 3 of the Status Register as a logic '0', the Sense A input is now returned to the sense mode and its function is once again identical to that of Sense B.

In some applications it may be desired to enable (or disable) the interrupt line and to set (or pulse) one or more flags with the same series of instructions. This can be done as follows:

```
1                         ;ENABLE INTERRUPT (IEN) AND SET FLAG
2        CSA              ;COPY STATUS TO ACCUMULATOR
3        ORI     009      ;ENABLE INTERRUPT AND SET FLAG 0
•
•
•
9                         ;DISABLE INTERRUPT (DINT) AND RESET FLAG
10       CSA              ;COPY STATUS TO ACCUMULATOR
11       ANI     OF6      ;DISABLE INTERRUPT AND RESET FLAG 0
12       CAS              ;COPY ACCUMULATOR TO STATUS
```

Bits 4 and 5 of the Status Register correspond, respectively, to the Sense A and Sense B inputs. If the Sense A line is high, a logic '1' is read into bit 4 of the Status Register, whereas if the Sense B line is high, status bit 5 is set to a logic '1'. The following series of instructions show one way to test the Sense A/Sense B status. The code in lines 1, 2, and 3 tests the status of Sense A (in the sense mode) and the code in lines 11, 12, and 13 tests the status of the Sense B input; the code in lines 21, 22, and 23 shows how both sense lines can be tested simultaneously.

Bits 6 and 7 of the Status Register provide arithmetic control; these control functions can be summarized as follows:

Bit 6 – Overflow (OV); this bit is set if an arithmetic overflow occurs during an add instruction (ADD, ADI, or ADE) or during a complement-and-add instruction (CAD, CAI, or CAE). Overflow is not affected by the decimal-add instructions (DAD, DAI, or DAE).

Bit 7 – Carry/Link (CY/L); this bit is set if a carry from the most significant bit occurs during an add, a complement-and-add, or a decimal-add instruction. The bit is also included in the Shift Right with Link (SRL) and the Rotate Right with Link (RRL) Instructions. CY/L is input as a carry into the bit 0 position of the add, complement-and-add, and decimal-add instructions.

SC/MP Pinouts

As shown in figure 1-3, SC/MP is housed in a 40-pin, dual-in-line package. Two of the pins are used for input power and two are used for timing; the remaining 36 pins are used for control, addressing, and data input/output functions. In a usual application, the 8 input/output pins (DB 00 through DB 07) are connected to a common data bus and the 12 address pins (AD 00 through AD 11) are connected to a common address bus. In conjunction with bus access and other appropriate control signals, three functions are implemented: (1) 8-bit data are input to SC/MP, (2) 8-bit data are output from SC/MP, and (3) address and status information are output from SC/MP. Timing detail and a description of each pinout are described in other documents; however, as an applications convenience, these functions are summarized, respectively, in figure 1-4 and table 1-1.

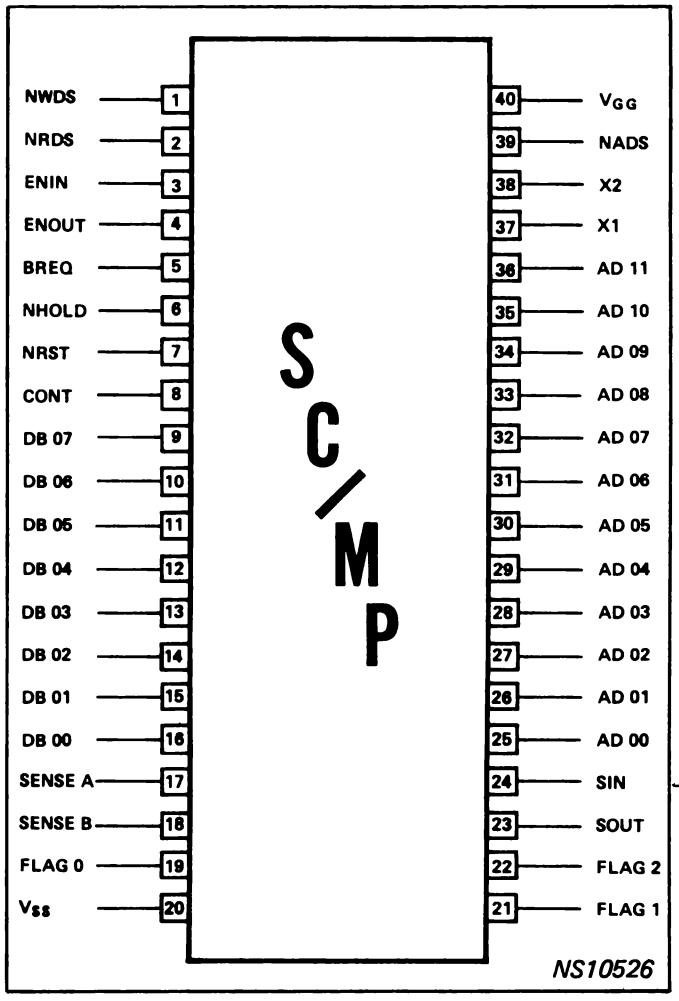


Figure 1–3. 40-Pin SC/MP Chip

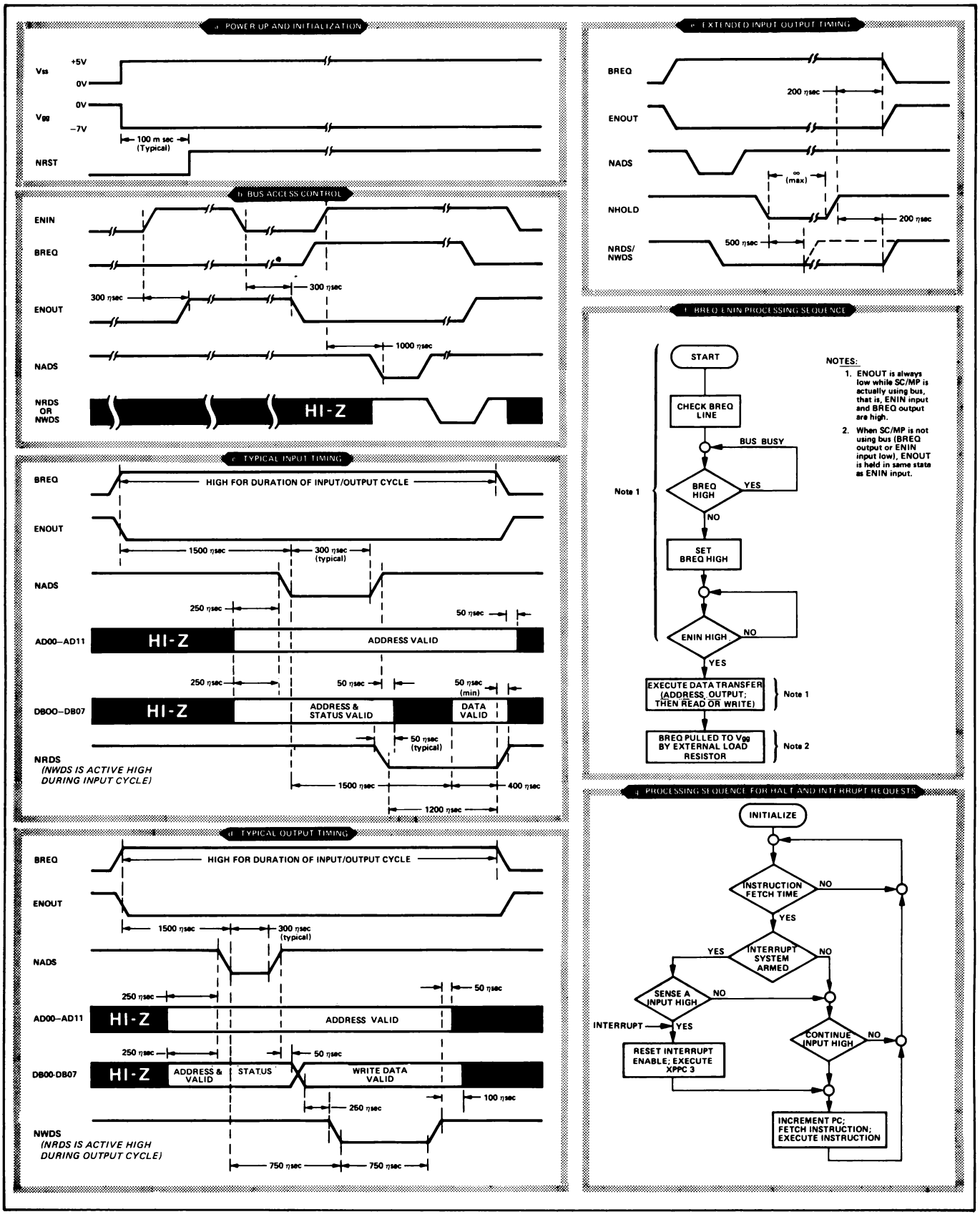


Figure 1–4. SC/MP Timing (Based on 1-MHz Crystal) and Processing Sequences

Table 1-1. Description of SC/MP Pinouts

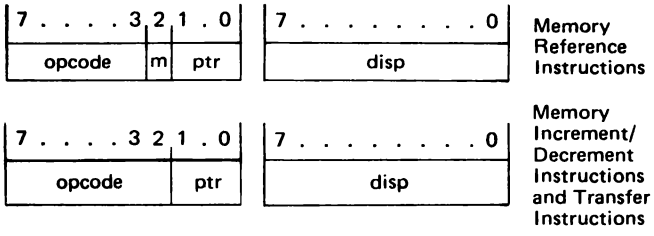
PIN	Basic Function	Design Considerations	PIN	Basic Function	Design Considerations
X1 37 X2 38	Timing	Connect capacitor between these pins for applications where timing is not critical; use crystal where timing is critical. (Refer to Appendix A for component characterization and the use of an external clock.)	NRST 7	Negative Reset	When this input is set low, in-process operations are aborted.
			NHOLD 6	Wait	In conjunction with CONT, the NHOLD input can be used to implement single cycle/single-instruction control of SC/MP — refer to figure 1-4e for extended input/output timing.
V_{SS} 20 V_{GG} 40	Power	V_{SS} = +5V (±5%) V_{GG} = −7V (±5%)			
			SIN 23 SOUT 24	Serial Input/ Output	When the SIO instruction is executed, the MSB of the input data is shifted into the MSB of the extension register, and the LSB is shifted from the E-Register to an output latch, that is, the contents of the register can be changed without affecting the state of the output latch.
Sense A 17 Sense B 18	External sensing and software-controlled interrupt	These TTL-level inputs are connected directly to bit positions 4 and 5 of the status register. Both bits can be copied from the status register to the accumulator but neither bit can be written into from the accumulator, that is, they are "read only" inputs. With the interrupt armed (bit 3 of status register set high), the Sense A pin becomes the interrupt input — see figure 1-4g for processing sequence of interrupt request and Appendix C for implementation detail of the interrupt system.	NADS 39	Negative Address Strobe	When low, indicates valid address and status outputs are present on the system buses. The NADS leading edge of the strobe can be used to externally latch input/output status and the four MSB of the 16-bit address; refer to figures 1-4c, 1-4d, and 1-4e for I/O timing of NADS.
Flag 0 19 Flag 1 21 Flag 2 22	External control of peripherals	Each flag output is TTL-compatible and can drive a 1.6-milliampere load. The flags are software-controlled and can be set or pulsed in a single or multiple sequence.			
			NRDS 2	Negative Read Strobe	A Tri-State output that, when low, indicates SC/MP is ready to accept data from the 8-bit input/output bus; as shown in figure 1-4c, data are input on the trailing edge of this strobe.
BREQ 5 ENIN 3 ENOUT 4	Bus-Access, DMA, and Multiprocessor Control	In simple stand-alone applications, BREQ can be connected to V_{GG} through a 6.8 kilohm resistor, ENOUT can be ignored, and ENIN can be connected to V_{SS} so that the SC/MP microprocessor has access to system buses whenever the BREQ pin is high. In systems that require bus-sharing, the common bus-request line is continually tested by each microprocessor; when the request line is low, system buses can be accessed, and if BREQ and ENIN are set high, bus access is granted.	NWDS 1	Negative Write Strobe	A Tri-State output that, when low, indicates output data from SC/MP is valid on 8-bit input/output bus; refer to figure 1-4d for output timing.
			DB 00-07 16...9	Input/Output data	At NADS time, I/O status and 4-MSB of 16-bit address are output from SC/MP; at NRDS time, data are input to SC/MP and, at NWDS time, data are output from SC/MP. Each pin is bidirectional and Tri-State.
CONT 8	Start/Stop	Permits suspension of operations without loss of internal status. Can be used with 'HALT' flag to implement a programmed halt; also, can be used with NHOLD (Wait) signal to implement single-cycle/single-instruction control of microprocessor.			
			AD 00–11 25 ... 36	Latched Address	At NADS time, the 12-bit latched address is valid and, as shown in figures 1-4c and 1-4d, a read *or* write function then is implemented.

*Refer to SC/MP data sheet for minimum/maximum values.

ADDRESSING CAPABILITIES AND INSTRUCTION SET OF SC/MP

Addressing

During execution, instructions and data defined in a program are stored into and loaded from specific memory locations, the Accumulator, or selected registers. Because SC/MP memory (read/write and read-only), and peripherals are on a common data bus, any instruction used to address memory may be used to address the peripherals. The formats of the instruction groups that reference memory are shown below.



Memory-reference instructions use the PC-relative, indexed, or auto-indexed methods of addressing memory. The memory-increment/decrement instructions and the transfer instructions use the PC-relative or indexed methods of addressing. The various methods of addressing memory and peripherals are shown below.

Type of Addressing	Operand Formats		
	m	ptr	disp
PC-relative	0	0	-128* to +127
Indexed	0	1, 2, or 3	-128* to +127
Immediate†	1	0	-128* to +127
Auto-indexed	1	1, 2, or 3	-128* to +127

* For PC-relative, indexed, and audo-indexed memory-reference instructions, another feature of the addressing architecture is that the contents of the Extension Register are substituted for the displacement if the instruction displacement equals -128 (X'80).

† Immediate addressing is an addressing format specific to immediate instructions.

NOTE

All arithmetic operations associated with address format affect only the 12 low-order address bits; no carry is provided to the 4 high-order bits. For systems employing memories of 4K or less, the high-order bits can be ignored, as they are set to 0000 following initialization. For systems employing larger memories, the high-order bits must be set to the starting address of the desired 4K block of memory. For example, when the 4 high-order bits are 0001_2, memory locations 1000_{16} - $1FFF_{16}$ are addressed; when 0010_2, memory locations 2000_{16} - $2FFF_{16}$ are addressed; and so forth.

PC-Relative Addressing

A PC-relative address is formed by adding the displacement value specified in the operand field of the instruction to the current contents of the Program Counter. The displacement is an 8-bit twos-complement number, so the range of the PC-relative addressing format is -127_{10} to $+127_{10}$ locations from the current contents of the Program Counter.

Immediate Addressing

Immediate addressing uses the value in the second byte of a double-byte instruction as the operand for the operation to be performed. For example, compare a Load Instruction (LD) to a Load Immediate Instruction (LDI). The Load Instruction uses the contents of the second byte of the instruction in computing the effective address of the data to be loaded. The Load Immediate Instruction uses the contents of the second byte as the data to be loaded.

Indexed Addressing

Indexed addressing enables the programmer to address any location in memory through the use of the Pointer Register and the displacement value of an instruction. When indexed addressing is specified in an instruction, the contents of the designated Pointer Register are added to the displacement to form the effective address. The contents of the Pointer Register are not modified by indexed addressing.

Auto-Indexed Addressing

Audo-indexed addressing provides the same capabilities as indexed addressing along with the ability to increment or decrement the contents of the designated Pointer Register by the value of the displacement. If the displacement is less than zero, the contents of the Pointer Register is decremented by the displacement before the contents of the effective address are fetched or stored. If the displacement is equal to or greater than zero, the contents of the Pointer Register are used as the effective address, and the contents of the Pointer Register are incremented by the displacement after the contents of the effective address are fetched or stored.

Instruction Set

The SC/MP instruction set provides the general-purpose user of microprocessors a powerful programming capability along with above-average flexibility and speed. The instruction set consists of 46 instructions, which comprise 8 general categories. A listing of the complete instruction set is provided in table 1-2; typical instruction execution times are given in table 1-3, and notations and symbols used as shorthand expressions of instruction capability are defined in table 1-4.

The instruction set includes both single-byte and double-byte instructions. A single-byte instruction consists of an 8-bit operation code that specifies an operation that SC/MP can execute without further reference to memory. A double-byte instruction consists of an 8-bit operation code and an 8-bit data or displacement field. When the second byte represents a data field, the data are processed by SC/MP during execution of the instruction, thereby eliminating the need for further memory references. When the second byte represents a displacement value, it is used to calculate a memory address that will be accessed (written into or read from) during execution of the instruction.

Figure 1-5 provides a flowchart that illustrates the execution sequence for the various classes of SC/MP instructions.

Table 1-2. SC/MP Instruction Summary

Mnemonic	Description	Object Format	Operation	Micro-Cycles
DOUBLE-BYTE INSTRUCTIONS				
	Memory Reference Instructions	7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0		
LD	Load	1 1 0 0 0 m ptr | disp	(AC)←(EA)	18
ST	Store	1 1 0 0 1 m ptr | disp	(EA)←(AC)	18
AND	AND	1 1 0 1 0 m ptr | disp	(AC)←(AC) (EA)	18
OR	OR	1 1 0 1 1 m ptr | disp	(AC)←(AC) V (EA)	18
XOR	Exclusive-OR	1 1 1 0 0 m ptr | disp	(AC)←(AC) V (EA)	18
DAD	Decimal Add	1 1 1 0 1 m ptr | disp	(AC)←(AC)$_{10}$ + (EA)$_{10}$ + (CY/L);(CY/L)	23
ADD	Add	1 1 1 1 0 m ptr | disp	(AC)←(AC) + (EA) + (CY/L);(CY/L),(OV)	19
CAD	Complement and Add	1 1 1 1 1 m ptr | disp	(AC)←(AC) + ~(EA) + (CY/L);(CY/L),(OV)	20
	Memory Increment/Decrement Instructions	7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0		
ILD	Increment and Load	1 0 1 0 1 0 ptr | disp	(AC), (EA)←(EA) + 1	22
DLD	Decrement and Load	1 0 1 1 1 0	(AC), (EA)←(EA) - 1	22
	Immediate Instructions	7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0		
LDI	Load Immediate	1 1 0 0 0 1 0 0 | data	(AC)←data	10
ANI	AND Immediate	1 1 0 1 0 1 0 0 | data	(AC)←(AC) data	10
ORI	OR Immediate	1 1 0 1 1 1 0 0 | data	(AC)←(AC) V data	10
XRI	Exclusive-OR Immediate	1 1 1 0 0 1 0 0 | data	(AC)←(AC) V data	10
DAI	Decimal Add Immediate	1 1 1 0 1 1 0 0 | data	(AC)←(AC)$_{10}$ + data$_{10}$ + (CY/L);(CY/L)	15
ADI	Add Immediate	1 1 1 1 0 1 0 0 | data	(AC)←(AC) + data + (CY/L);(CY/L),(OV)	11
CAI	Complement and Add Immediate	1 1 1 1 1 1 0 0 | data	(AC)←(AC) + ~data + (CY/L);(CY/L),(OV)	12
	Transfer Instructions	7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0		
JMP	Jump	1 0 0 1 0 0 ptr | disp	(PC)←EA	11
JP	Jump if Positive	1 0 0 1 0 1 ptr | disp	If (AC) ≥ 0, (PC)←EA	9, 11
JZ	Jump if Zero	1 0 0 1 1 0 ptr | disp	If (AC) = 0, (PC)←EA	9, 11
JNZ	Jump if Not Zero	1 0 0 1 1 1 ptr | disp	If (AC) ≠ 0, (PC)←EA	9, 11
	Double-Byte Miscellaneous Instructions	7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0		
DLY	Delay	1 0 0 0 1 1 1 1 | data	count AC to -1, delay = 13 + 2(AC) + 2 disp + 2^9 disp microcycles	13 to 131, 593

Table 1-2 (Concluded)

Mnemonic	Description	Object Format	Operation	Micro-Cycles
SINGLE-BYTE INSTRUCTIONS				
	Extension Register Instructions	7 6 5 4 3 2 1 0		
LDE	Load AC from Extension	0 1 0 0 0 0 0 0	(AC)←(E)	6
XAE	Exchange AC and Extension	0 0 0 0 0 0 0 1	(AC)↔(E)	7
ANE	AND Extension	0 1 0 1 0 0 0 0	(AC)←(AC) (E)	6
ORE	OR Extension	0 1 0 1 1 0 0 0	(AC)←(AC) V (E)	6
XRE	Exclusive-OR Extension	0 1 1 0 0 0 0 0	(AC)←(AC) V (E)	6
DAE	Decimal Add Extension	0 1 1 0 1 0 0 0	$(AC)\leftarrow(AC)_{10} + (E)_{10} + (CY/L);(CY/L)$	11
ADE	Add Extension	0 1 1 1 0 0 0 0	(AC)←(AC) + (E) + (CY/L);(CY/L),(OV)	7
CAE	Complement and Add Extension	0 1 1 1 1 0 0 0	(AC)←(AC) + ~(E) + (CY/L);(CY/L),(OV)	8
	Pointer Register Move Instructions	7 6 5 4 3 2 1 0		
XPAL	Exchange Pointer Low	0 0 1 1 0 0 ptr	$(AC)\leftrightarrow(PTR_{7:0})$	8
XPAH	Exchange Pointer High	0 0 1 1 0 1 ptr	$(AC)\leftrightarrow(PTR_{15:8})$	8
XPPC	Exchange Pointer with PC	0 0 1 1 1 1 ptr	(PC)↔(PTR)	7
	Shift, Rotate, Serial I/O Instructions	7 6 5 4 3 2 1 0		
SIO	Serial Input/Output	0 0 0 1 1 0 0 1	$(E_i)\rightarrow(E_{i-1})$, SIN→$(E_7)$, (E_0)→SOUT	5
SR	Shift Right	0 0 0 1 1 1 0 0	$(AC_i)\rightarrow(AC_{i-1})$, 0→$(AC_7)$	5
SRL	Shift Right with Link	0 0 0 1 1 1 0 1	$(AC_i)\rightarrow(AC_{i-1})$, (CY/L)→$(AC_7)$	5
RR	Rotate Right	0 0 0 1 1 1 1 0	$(AC_i)\rightarrow(AC_{i-1})$, $(AC_0)\rightarrow(AC_7)$	5
RRL	Rotate Right with Link	0 0 0 1 1 1 1 1	$(AC_i)\rightarrow(AC_{i-1})$, $(AC_0)\rightarrow(CY/L)\rightarrow(AC_7)$	5
	Single-Byte Miscellaneous Instructions	7 6 5 4 3 2 1 0		
HALT	Halt	0 0 0 0 0 0 0 0	Pulse H-flag	8
CCL	Clear Carry/Link	0 0 0 0 0 0 1 0	(CY/L)←0	5
SCL	Set Carry/Link	0 0 0 0 0 0 1 1	(CY/L)←1	5
DINT	Disable Interrupt	0 0 0 0 0 1 0 0	(IE)←0	6
IEN	Enable Interrupt	0 0 0 0 0 1 0 1	(IE)←1	6
CSA	Copy Status to AC	0 0 0 0 0 1 1 0	(AC)←(SR)	5
CAS	Copy AC to Status	0 0 0 0 0 1 1 1	(SR)←(AC)	6
NOP	No Operation	0 0 0 0 1 0 0 0	None	5

Table 1-3. Instruction Execution Time

Instruction	Read Cycles	Write Cycles	Total Microcycles
ADD	3	0	19
ADE	1	0	7
ADI	2	0	11
AND	3	0	18
ANE	1	0	6
ANI	2	0	10
CAD	3	0	20
CAE	1	0	8
CAI	2	0	12
CAS	1	0	6
CCL	1	0	5
CSA	1	0	5
DAD	3	0	23
DAE	1	0	11
DAI	2	0	15
DINT	1	0	6
DLD	3	1	22
DLY	2	0	13 - 131593
HALT	2	0	8
IEN	1	0	6
ILD	3	1	22
JMP	2	0	11
JNZ	2	0	9, 11 for Jump

Instruction	Read Cycles	Write Cycles	Total Microcycles
JP	2	0	9, 11 for Jump
JZ	2	0	9, 11 for Jump
LD	3	0	18
LDE	1	0	6
LDI	2	0	10
NOP	1	0	5
OR	3	0	18
ORE	1	0	6
ORI	2	0	10
RR	1	0	5
RRL	1	0	5
SCL	1	0	5
SIO	1	0	5
SR	1	0	5
SRL	1	0	5
ST	2	1	18
XAE	1	0	7
XOR	3	0	18
XPAH	1	0	8
XPAL	1	0	8
XPPC	1	0	7
XRE	1	0	6
XRI	2	0	10

Table 1-4. Symbols and Notations Used to Express Instruction Execution

Symbol and Notation	Meaning
AC	8-bit Accumulator.
CY/L	Carry/Link Flag in the Status Register.
data	8-bit immediate data field. Data may represent a signed or unsigned twos complement number or two binary-coded-decimal (BCD) numbers.
disp	Displacement; represents a signed 8-bit address modifier in a memory reference, memory increment/decrement, or transfer instruction.
EA	Effective Address as specified by the instruction.
E	Extension Register; provides for temporary storage, variable displacements and separate serial input/output port.
i	Represents a bit in one of the bit positions, 7 through 1, of the Accumulator or the Extension Register.
IE	Interrupt Enable Flag.
m	Mode bit, used in memory reference instructions. Blank parameter sets m = 0, @ sets m = 1.
OV	Overflow Flag in the Status Register.
PC	Program Counter (Pointer Register 0); during address formation, PC points to the last byte of the instruction being executed.
ptr	Pointer Register (ptr = 0 through 3). The register specified in byte 1 of the instruction.
$ptr_{n:m}$	Pointer register bits; n:m = 7 through 0 or 15 through 8.
SIN	Serial Input pin.
SOUT	Serial Output pin.
SR	8-bit Status Register.
( )	Means "contents of." For example, (EA) is contents of Effective Address.
[ ]	Means optional field in the assembler instruction format.
~	Ones complement of value to right of ~.
→	Means "replaces."
←	Means "is replaced by."
↔	Means "exchange."
@	When used in the operand field of the instruction, sets the mode bit (m) to 1 for auto-incrementing/auto-decrementing indexing.
10^+	Modulo 10 addition.
Λ	AND operation.
V	Inclusive-OR operation.
⊻	Exclusive-OR operation.
⩾	Greater than or equal to.
=	Equals.
≠	Does not equal.

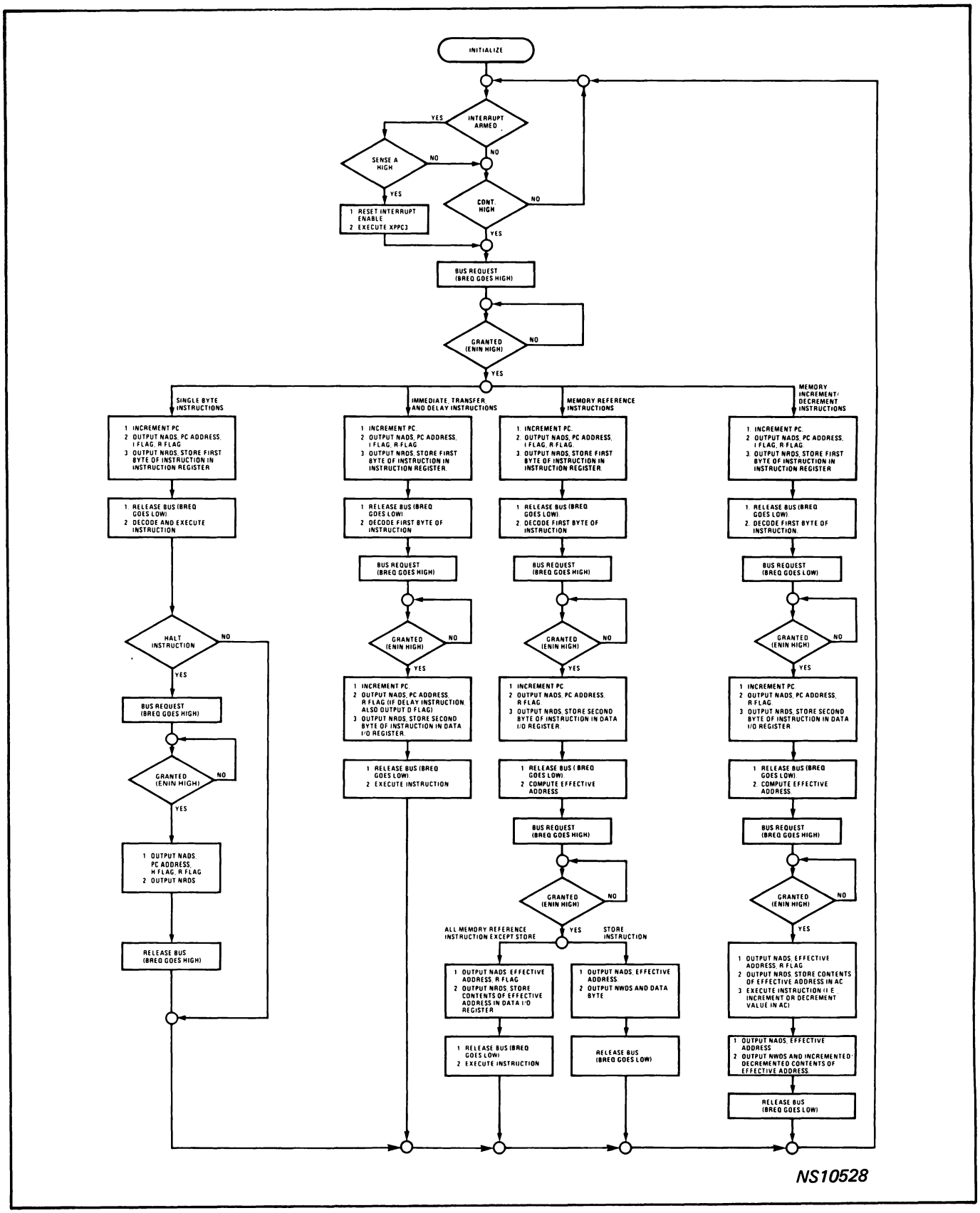


Figure 1–5. SC/MP Program Execution

IMPLEMENTING A MINIMUM (LOW-COST) SC/MP SYSTEM

The preceding information can be summarized to say that SC/MP is a complete CPU without an on-chip memory. The stand-alone CPU is an advantage because memory requirements are frequently an applications function and, with SC/MP, the user can select the memory that is best suited for a particular use. With the addition of memory, a SC/MP-based low-cost system can be developed easily; such a system is shown in figure 1-6. This 2-chip system (SC/MP plus memory) is a minimal configuration that may be used in some applications, a few of which are described later in this book. This particular system can be used in many security-entry applications – to open a door or to gain coded-access to files, safes, and so forth. Operation of the system is summarized as follows:

- 8-bit employee-access code is input one bit at a time via the DATA SET and DATA ENTER switches.
- Entered code is compared to a prestored reference in memory.
- If code is valid, ACCEPT indicator lights and LOCK opens; if code is invalid, REJECT indicator lights and system is reset to zero.

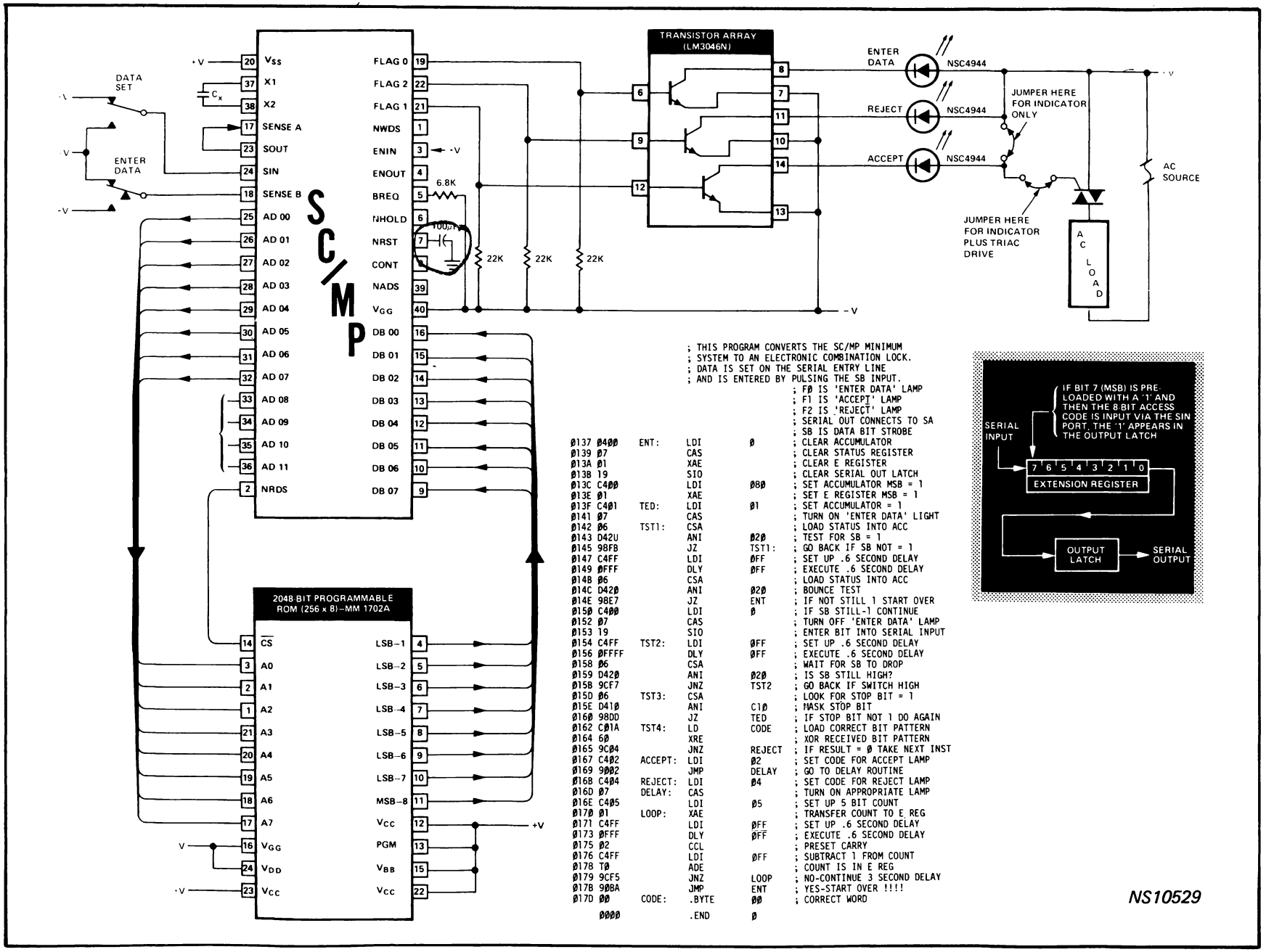
The system must indicate to the employee when to enter data; this is done via flag 0 (F0) and its associated ENTER DATA indicator. When the indicator is lit, the system is ready to accept data, and when the indicator is extinguished, the system is reading the status of the DATA SET switch. Flags 1 and 2, respectively, are used to specify code-accept and code-reject conditions. In keeping with the theme of minimum cost, LEDs are used for the enter-data, code-accept, and code-reject indicators; since the devices are inherently current-limiting, no surge resistor is required in the base of the transistor and no series resistor is required in the collector circuit.

One other function is required for system operation – some way of determining when 8 bits (the complete access code) are entered. Referring to the Extension Register and output latch shown as a screened inset in figure 1-6, it is seen that if the most significant bit (bit 7) is preloaded with a '1' and then the 8-bit access code is inputted, the preloaded value of '1' is shifted into the output latch. If the latch is preloaded with a '0' (which is done by the software) and is connected to Sense A of SC/MP, the system can determine when the code is complete (8 bits are entered) simply by checking Sense A after the entry of each bit. When Sense A is equal to '1', the input code is accepted, compared with a prestored reference, and if the two codes are in agreement, the lock is opened. If the two codes do not agree, the REJECT indicator lights.

Reviewing the operation, the user looks to see if the ENTER DATA indicator is lit – if it is lit, the least significant bit of the authorization code is selected with the DATA SET switch and entered with the ENTER DATA pushbutton. While this bit is being read, the ENTER DATA indicator extinguishes and then comes back on; at this time, the set-data/enter-data sequence is repeated for the second bit of the access code. At the end of the eighth entry, a '1' appears at the serial output port (Sense A). The Sense A line is tested at the completion of each entry, and when the output latch is equal to '1', the code is complete and is checked against a prestored reference for validity. The program is setup such that each entry is timed for 20 seconds; if the next bit is not entered within this time period, the system resets to all zeroes. This prevents partial entries from disabling the system.

EXPANDING THE SECURITY-ENTRY SYSTEM

In the preceding system, coded access is provided for a "single" lock, drawer, or door. The same basic system can be expanded to serve several access points; such a system is shown in figure 1-7. The added component is a CMOS 1-of-10 line decoder with a high-current pullup capability at the output. As shown, three inputs to the decoder are supplied by flags 0, 1, and 2; the fourth input is supplied by the inverted output of the SOUT line. Code-entry and code-reject features are similar to those shown in figure 1-6. As long as the serial output (SOUT) line is a logic '0' and every flag is a logic '0', the DATA ENTRY line (pin 9) is selected. When flag 0 is high and other inputs are the same as before, the REST (ready for data) line is selected.




```
; THIS PROGRAM CONVERTS THE SC/MP MINIMUM
; SYSTEM TO AN ELECTRONIC COMBINATION LOCK.
; DATA IS SET ON THE SERIAL ENTRY LINE
; AND IS ENTERED BY PULSING THE SB INPUT.
                                ; FØ IS 'ENTER DATA' LAMP
                                ; F1 IS 'ACCEPT' LAMP
                                ; F2 IS 'REJECT' LAMP
                                ; SERIAL OUT CONNECTS TO SA
                                ; SB IS DATA BIT STROBE
Ø137 Ø4ØØ   ENT:    LDI   Ø     ; CLEAR ACCUMULATOR
Ø139 Ø7             CAS         ; CLEAR STATUS REGISTER
Ø13A Ø1             XAE         ; CLEAR E REGISTER
Ø13B 19             SIO         ; CLEAR SERIAL OUT LATCH
Ø13C C4ØØ           LDI   Ø8Ø   ; SET ACCUMULATOR MSB = 1
Ø13E Ø1             XAE         ; SET E REGISTER MSB = 1
Ø13F C4Ø1   TED:    LDI   Ø1    ; SET ACCUMULATOR = 1
Ø141 Ø7             CAS         ; TURN ON 'ENTER DATA' LIGHT
Ø142 Ø6     TST1:   CSA         ; LOAD STATUS INTO ACC
Ø143 D42U           ANI   Ø2Ø   ; TEST FOR SB = 1
Ø145 98FB           JZ    TST1: ; GO BACK IF SB NOT = 1
Ø147 C4FF           LDI   ØFF   ; SET UP .6 SECOND DELAY
Ø149 ØFFF           DLY   ØFF   ; EXECUTE .6 SECOND DELAY
Ø14B Ø6             CSA         ; LOAD STATUS INTO ACC
Ø14C D42Ø           ANI   Ø2Ø   ; BOUNCE TEST
Ø14E 98E7           JZ    ENT   ; IF NOT STILL 1 START OVER
Ø15Ø C4ØØ           LDI   Ø     ; IF SB STILL-1 CONTINUE
Ø152 Ø7             CAS         ; TURN OFF 'ENTER DATA' LAMP
Ø153 19             SIO         ; ENTER BIT INTO SERIAL INPUT
Ø154 C4FF   TST2:   LDI   ØFF   ; SET UP .6 SECOND DELAY
Ø156 ØFFFF          DLY   ØFF   ; EXECUTE .6 SECOND DELAY
Ø158 Ø6             CSA         ; WAIT FOR SB TO DROP
Ø159 D42Ø           ANI   Ø2Ø   ; IS SB STILL HIGH?
Ø15B 9CF7           JNZ   TST2  ; GO BACK IF SWITCH HIGH
Ø15D Ø6     TST3:   CSA         ; LOOK FOR STOP BIT = 1
Ø15E D41Ø           ANI   C1Ø   ; MASK STOP BIT
Ø16Ø 98DD           JZ    TED   ; IF STOP BIT NOT 1 DO AGAIN
Ø162 CØ1A   TST4:   LD    CODE  ; LOAD CORRECT BIT PATTERN
Ø164 6Ø             XRE         ; XOR RECEIVED BIT PATTERN
Ø165 9C04           JNZ   REJECT ; IF RESULT = Ø TAKE NEXT INST
Ø167 C4Ø2   ACCEPT: LDI   Ø2    ; SET CODE FOR ACCEPT LAMP
Ø169 9ØØ2           JMP   DELAY ; GO TO DELAY ROUTINE
Ø16B C4Ø4   REJECT: LDI   Ø4    ; SET CODE FOR REJECT LAMP
Ø16D Ø7     DELAY:  CAS         ; TURN ON APPROPRIATE LAMP
Ø16E C4Ø5           LDI   Ø5    ; SET UP 5 BIT COUNT
Ø17Ø Ø1     LOOP:   XAE         ; TRANSFER COUNT TO E REG
Ø171 C4FF           LDI   ØFF   ; SET UP .6 SECOND DELAY
Ø173 ØFFF           DLY   ØFF   ; EXECUTE .6 SECOND DELAY
Ø175 Ø2             CCL         ; PRESET CARRY
Ø176 C4FF           LDI   ØFF   ; SUBTRACT 1 FROM COUNT
Ø178 TØ             ADE         ; COUNT IS IN E REG
Ø179 9CF5           JNZ   LOOP  ; NO-CONTINUE 3 SECOND DELAY
Ø17B 9ØBA           JMP   ENT   ; YES-START OVER !!!!
Ø17D ØØ     CODE:   .BYTE ØØ    ; CORRECT WORD
     ØØØØ           .END  Ø
```

NS10529

Figure 1–6. Minimum Security System Using SC/MP and PROM

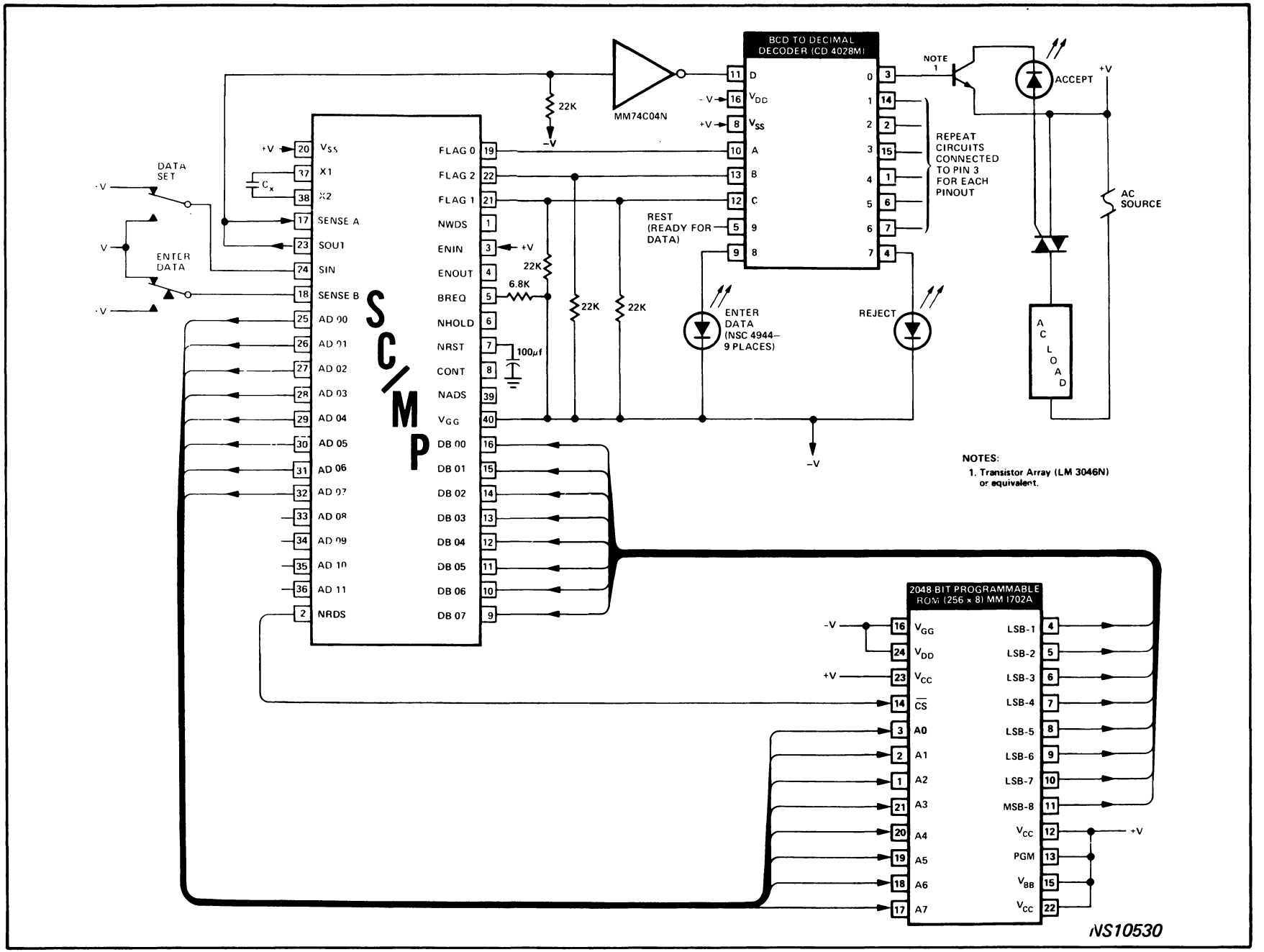


Figure 1–7. Expanded Security System Using SC/MP, PROM, a 4-by-10 Decoder, and Miscellaneous Components

BUFFERING AND INTERFACE CHARACTERISTICS OF SC/MP

In any application, buffering and interfacing capabilities of SC/MP are important design considerations; the following sections are addressed to these parameters.

TTL/MOS Interfaces

From an overall interface point-of-view, the current and voltage characteristics of SC/MP are summarized as follows.

- Except for BREQ (pin 5) and X1/X2 (pins 37/38), all *input* pins typically present a 1.4-milliampere TTL load to any driving device.
- Except for BREQ, all *output* pins can drive a 1.6-milliampere TTL load.
- With V_{SS} = +5V and V_{GG} = -7V, SC/MP is voltage-compatible with TTL devices; thus, it can interface directly with TTL logic. Direct interface with 5-volt MOS logic also can be implemented.

NOTE

Refer to SC/MP Data Sheet for minimum/maximum input/output specifications.

To minimize buffering requirements, it is advantageous to use MOS, or Low-Power TTL devices for direct interface with SC/MP. Usually, the low-power devices present much less than a 1.6-milliampere load and their propagation delay times compare favorably with the timing parameters of SC/MP.

Buffering SC/MP

As indicated in the preceding paragraph, SC/MP can generally interface with MOS and low-power TTL circuits without the use of buffers. In applications where the SC/MP drives more than one TTL load, buffering is required. One method of buffering SC/MP outputs is shown in figure 1-8. To minimize component count and to conserve power, low-power Schottky TRI-STATE® octal buffers are used; in addition to data-, address-, and control-line buffering, the high-order address bits (AD 12-AD 15) are latched to support large memories and/or a full complement of input/output peripherals. Each buffered output line can drive 10 or more TTL loads – approximately 16 milliamperes. To determine if buffered or direct-to-chip connections are required for a particular application, the user must consider carefully such system parameters as overall loading, sink-current capabilities of SC/MP, duty cycle, peak power, and so on. Refer to the SC/MP Data Sheet for parametric specifications.

The power-up and initialization circuit shown in figure 1-8 is designed to accommodate any 'clock' technique (appendix A) that is used with SC/MP. An RC network can be used for the NRST input but does not provide timing that is as precise and probably not as reliable as would be provided by the Schmitt Trigger. (Note: If a manual reset is desired, a switch can be connected from the input of the Schmitt trigger to ground.)

® Registered trademark of the National Semiconductor Corporation

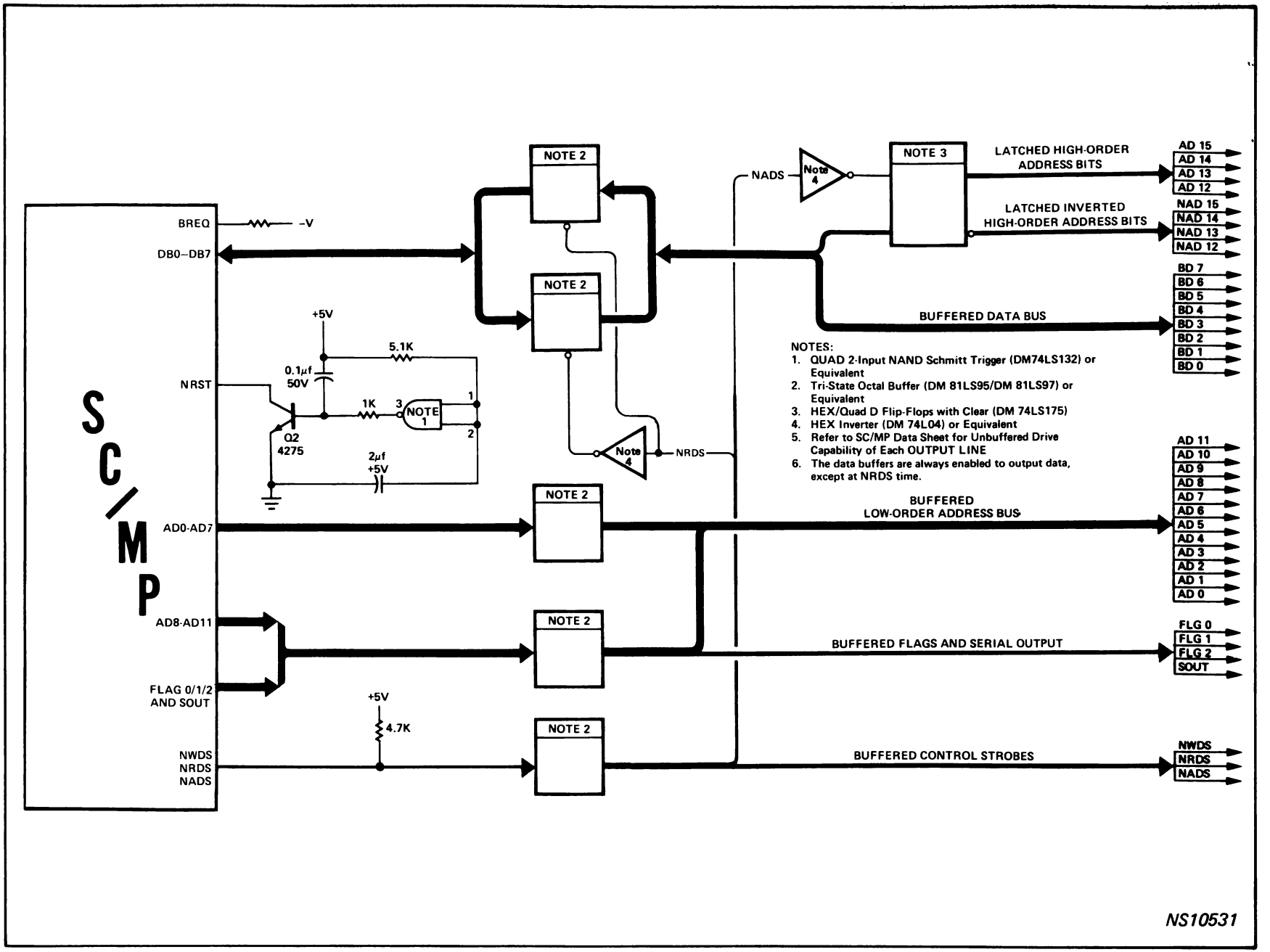


Figure 1–8. One Method of Buffering Data , Address , and Control Lines of SC/MP

TRI-STATE Considerations

In figures 1-2 and 1-4, the 12-bit latched address port (AD 00-AD 11), the 8-bit input/output port (DB 00-DB 07), and the read (NRDS)/Write (NWDS) strobes have three separate output states – a TTL logic '0', a TTL logic '1', and a high-impedance (HI-Z) (TRI-STATE) output. The '0' and '1' states are self-explanatory; an examination of figure 1-9 will show why the third (HI-Z) state is required. As shown, the 8-bit input/output bus is bidirectional; that is, during a read cycle, the internal receivers of SC/MP are connected to the bus and, during a write cycle, the SC/MP drivers are connected to the bus. If the TRI-STATE device (shown shaded in figure 1-9) is removed, the TTL outputs of SC/MP and of the TTL device are connected directly to the bus and both will attempt to drive it. Generally, the drive capability of the TTL device is the greater of the two; thus, it will prevail and system control by SC/MP is lost. With the TRI-STATE device connected, the TTL device is effectively disconnected from the bus; however, at read strobe (NRDS) time, the output of the buffer is enabled and the drivers of SC/MP are disabled. Accordingly, the SC/MP receivers read whatever is put on the bus by the buffer.

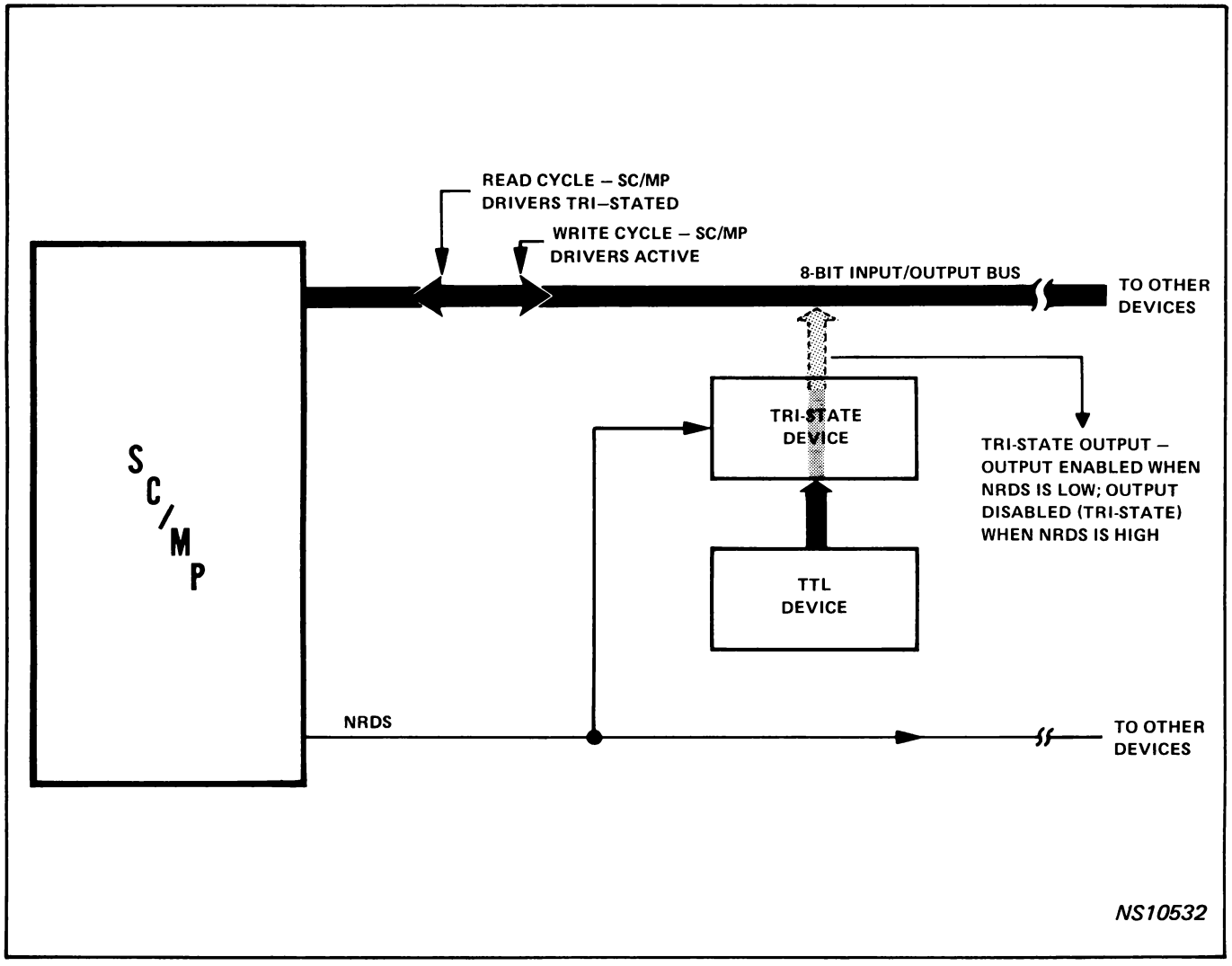


Figure 1–9. TRI-STATE Bus Interface

Figure 1-10 shows a method of implementing TRI-STATE control. With both TRI-STATE ENABLE lines high, both the memory and peripheral devices are effectively disconnected from the data bus; that is, the output drivers of each device are in the high-impedance mode. At read strobe time, one of the devices is selected; if address bit (AD 08) is low (inverted high), the TRI-STATE ENABLE line of ROM/PROM is driven low and this memory device is selected as the bus driver. Conversely, if AD 08 is high, the other select line is enabled and the peripheral is selected to drive the bus. Since each device is selected by a discrete address, the processor has absolute control over each "receiver" or "transmitter" connected to the bus. (Note: The basic address-decoding scheme shown in figure 1-10 can be expanded to serve small-memory systems (up to 4K) that require multiple read/write peripherals; refer to appendix B for further detail.)

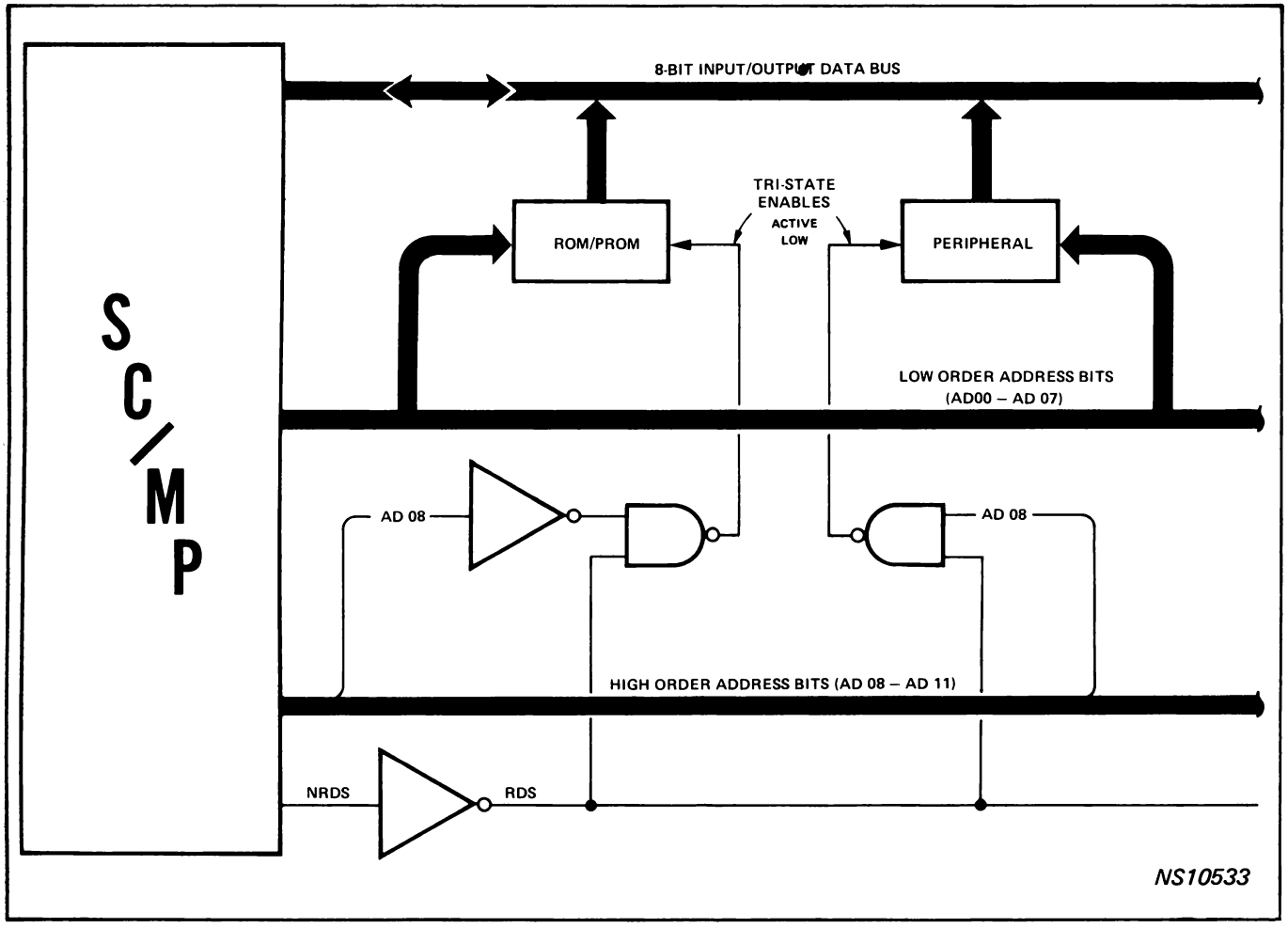


Figure 1–10. One Method of TRI-STATE Control

Chapter 2

CONCEPTS AND PRINCIPLES OF SC/MP INTERFACING

In the preceding section of this manual, SC/MP is defined in terms of general-purpose applications–timing, loading, peripheral interfacing, software manipulation, and so on. In this section, these SC/MP parameters are brought together to explain *how to hook-up, how to implement,* and *how to control* a variety of functional SC/MP-based applications. For current convenience and future add-on flexibility, the applications are organized by class.

GENERAL CONCEPTS OF A/D AND D/A CONVERTERS

Generalized concepts of how a SC/MP-based system can be used in a general-purpose analog-to-digital converter are shown in figure 2C1–1. The analog source can be any device capable of producing a current or a low-voltage output over a predetermined range. Under program control and system timing parameters, the analog source is sampled by the Analog-to-Digital Converter and the resulting output is a digital word with 8-bit resolution. The digital word is stored in RAM where, under program direction, it can serve a number of functions. For example, the converted data can be compared to a previously stored reference value; thus, in a quality-control configuration, for example, a pass or a reject decision can be made. As another example, the difference between the input data and the stored reference can be treated as an error signal, and when reconverted to its analog equivalent (shown with broken-line blocks and lines), it can be used in applications that require coordinate control. The output data also can be listed alphanumerically for statistical studies in applications where time is plotted against some other variable. In subsequent sections, some typical analog-to-digital conversion schemes are described. Refer to chapter 1 of this manual for timing information, pinout descriptions, and interfacing detail of the SC/MP chip.

SINGLE-INPUT ANALOG-TO-DIGITAL CONVERTER

General Description

The SC/MP-controlled analog-to-digital converter shown in figure 2C1–2 is well-suited to applications such as simple machine control, single-parameter testing, data acquisition, and other single-input functions. Requiring few components, minimum memory, and a simple program, the single-input analog-to-digital converter is easy and inexpensive to implement. Operating principles of analog-to-digital converters and logic circuits used in this application are well-defined in textbooks and industrial manuals; thus, the following descriptions are aimed primarily at the functional interfaces, user-supplied system parameters, and control and supervision of the software.

System Operation

Other than supply voltages, the single-input converter system requires a start pulse, clock pulses, an output-enable gate, and, of course, an analog input. Providing that a valid

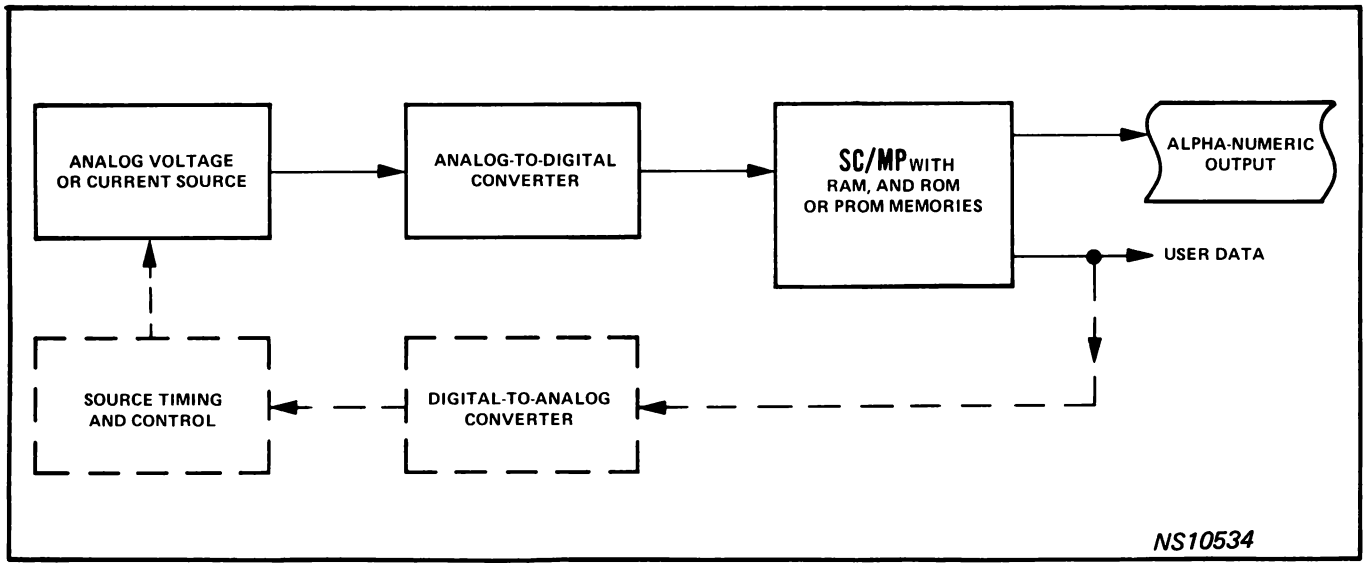


Figure 2C1–1. Principles of Analog-to-Digital/Digital-to-Analog Conversion

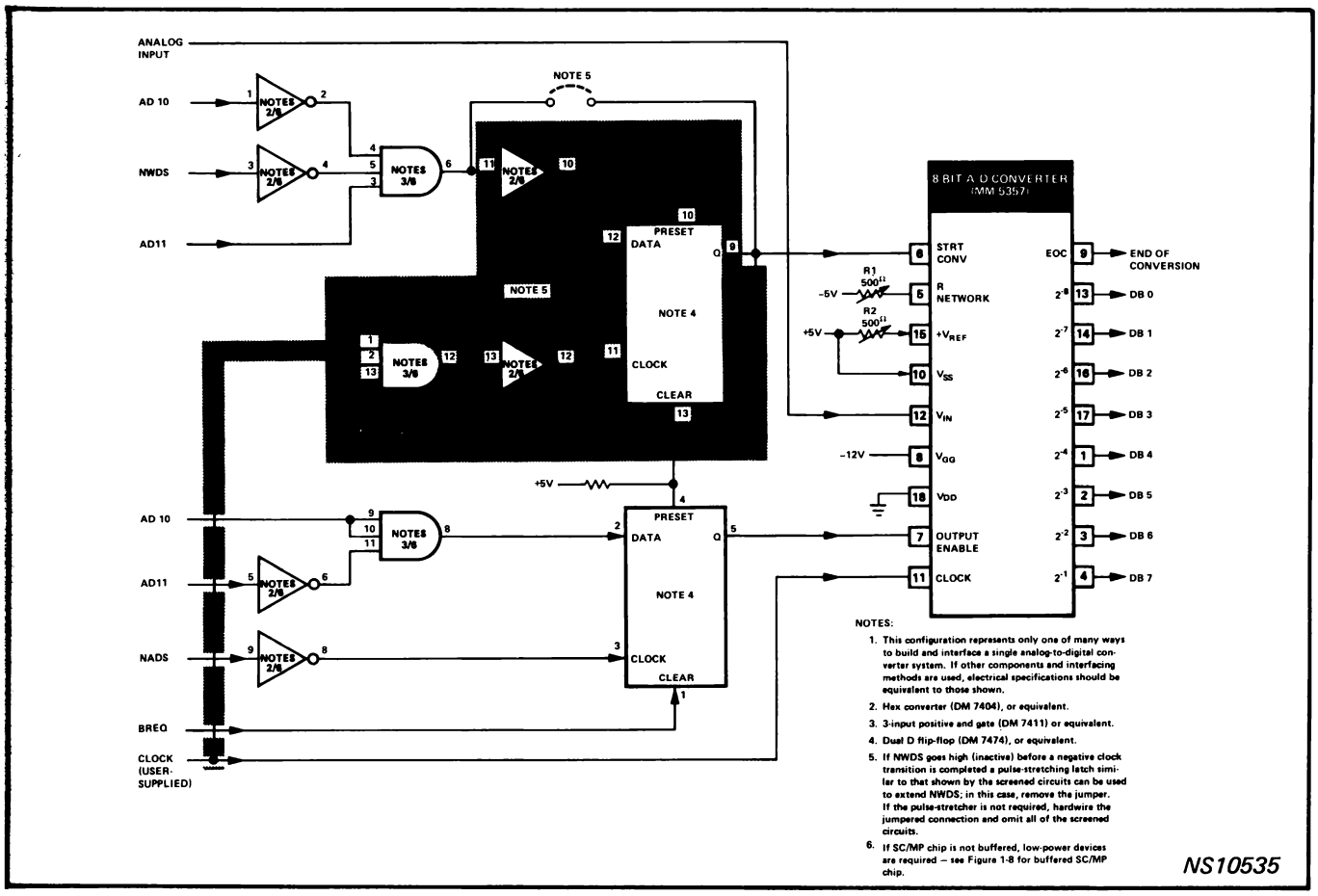


Figure 2C1–2. Single-Converter Analog-to-Digital System

start-converter pulse is present (logic 1 at pin 6), the conversion starts on the trailing edge (high-to-low transition) of the first clock pulse and continues for 40 clock cycles. When the conversion is completed, an 8-bit digital word is loaded into an output latch and an end-of-conversion (EOC) logic level is generated. The binary output (DB0 through DB7) is TRI-STATE to permit the use of common bus lines. When a valid address is received, an output-enable signal is generated; at this time, the digital output enters the accumulator of SC/MP and subsequently is stored in a designated memory location. Valid data are held in the output latch from the end of one conversion to the end of the next; thus, data transfers to memory can be implemented asynchronously.

Timing for one conversion cycle is summarized in figure 2C1–3. As shown, the conversion begins on the trailing edge of the clock pulse; thus, the 'start converter' gate must be at least as wide and preferably somewhat wider than one clock cycle. Referring to figure 2C1–3, it is seen that the width of the 'start' gate (STRT CONV) is determined by the write strobe (NWDS) of SC/MP. If the analog-to-digital clock is slower than NWDS, a pulse-stretching circuit similar to that noted in figure 2C1–2 is required; the unused flip-flop in the DM7474 package is available for this purpose. In figure 2C1–2, a starting address of X'0800 has been arbitrarily chosen—any other nonconflicting address could be used.

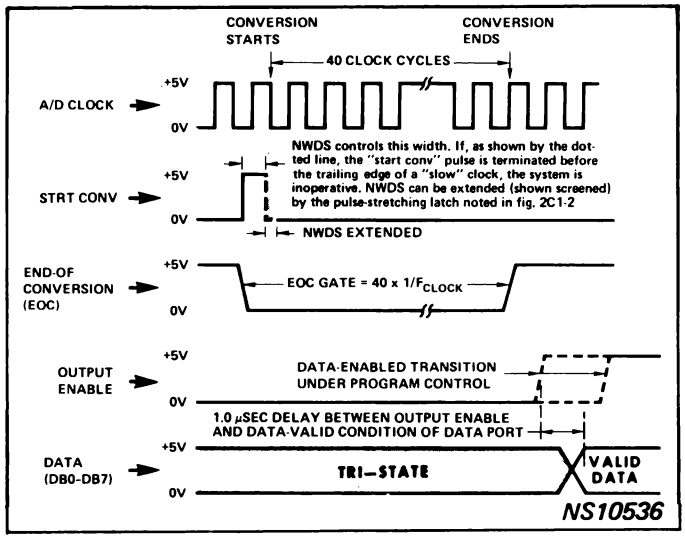


Figure 2C1–3. Timing Summary for Single-Converter System

When the conversion cycle starts, the analog input is admitted at pin 12 of the converter chip and the end-of-conversion (EOC) gate is set low. During the next 40 clock cycles, the input is sampled continuously and, via a process of successive approximation, the analog signal is converted into an 8-bit digital word. At the end of the fortieth clock cycle, the conversion is complete and two things happen—the EOC gate is set high and the digital word is loaded into an output latch on the converter chip.

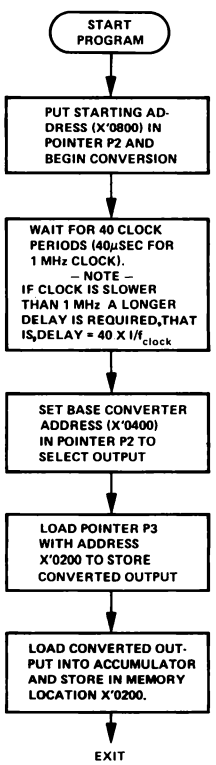
When the output is enabled (Output Enable set high), the latched data (DB0 through DB7) are available at pins 13, 14, 16, 17, and 1 through 4, respectively. In likeness to the STRT CONV gate, the Output Enable gate is generated via an arbitrary address—in this case, 0400_{16}. The Output Enable gate is synchronized by the address strobe (NADS) from SC/MP, and the gate remains active high until the bus request line (BREQ) is released (goes low) by the microprocessor. The output control functions are under software control and, as shown in figure 2C1-2, are implemented by a flip-flop.

System Adjustments

With supply voltages as shown, the analog-to-digital converter in figure 2C1-2 is designed to operate over an input range of 10 volts (±5 volts). Two adjustments (R1 and R2) are provided to optimize conversion accuracy. Variable resistor R1 is the zero adjustment, and for a 10-volt scale, it is set for a transition from '11111111' to '11111110' to occur at 19.53 millivolts (that is, one-half of the least significant bit value). If the voltage difference between pins 5 and 15 is more than 10 volts, then the half-bit zero-adjustment value is obtained by dividing 528 (the number of half-bit values) into the difference voltage. For instance, if the voltage between pins 5 and 15 is 10.24 volts, R1 is adjusted for 20 millivolts at the transition point—'11111111' to '11111110'. Resistor R2 is the full-scale adjustment, and for a 10-volt scale, it is set for the transition from '00000001' to '00000000' to occur at 58.6 millivolts (that is, one and one-half times the least significant bit value). Again, if the difference voltage is 10.24 volts, R2 is adjusted for 60 millivolts at the transition point—'00000001' to '00000000'.

Software Considerations

The flowchart and program listing in figure 2C1-4 shows how the single-input analog-to-digital converter system (figure 2C1-2) can be software-controlled to provide the functions described under system operation. Referring to figures 2C1-2 and 2C1-4, the software-hardware interface can be summarized as follows.




```
1          TITLE  SCMP, SINGLE A/D CONVERTER PROGRAM
2          ØØØ3  P3=3
3          ØØØ2  P2=2
4          ØØØØ  CONVØ=Ø
5  ØØØØ    Ø8              NOP
6  ØØØ1    C4ØØ  START:    LDI   X'ØØ      ;LOAD P2 WITH ADRS OF CONVERTER
7  ØØØ3    32              XPAL  P2
8  ØØØ4    C4Ø8            LDI   X'Ø8
9  ØØØ6    36              XPAH  P2
1Ø ØØØ7    CAØØ            ST    CONVØ(P2) ;START CONVERTER Ø
11 ØØØ9    Ø8              NOP             ;IF REQUIRED DELAY INTRODUCED TO
12 ØØØA    Ø8              NOP             ; COMPLETE THE CONVERSION
13 ØØØB    Ø8              NOP
14 ØØØC    Ø8              NOP
15 ØØØD    C4Ø4  ACCEPT:   LDI   X'Ø4      ;LD P2 WITH BASE CONV ADRS
16 ØØØF    36              XPAH  P2
17 ØØ1Ø    C4Ø2            LDI   X'Ø2      ;LD P3 WITH MEM ADDRS FOR
18 ØØ12    37              XPAH  P3        ;CONVERTED DIGITAL OUTPUT
19 ØØ13    C4ØØ            LDI   X'ØØ
2Ø ØØ15    33              XPAL  P3
21 ØØ16    C2ØØ            LD    CONVØ(P2) ;READ IN CONVERTED OUTPUT
22 ØØ18    C8ØØ  STORE:    ST    (P3)      ;STORE CONVERTED OUTPUT
                                             ;IN LOCATION 2ØØ
```

NS10537

Figure 2C1–4. Flowchart and Program Listing for Single-Converter System

At the start of the program, P2 is loaded with the starting address, X'0800 (lines 6 through 9), a STore Instruction is executed, and the 12-line address port of SC/MP is latched at X'0800 for the remainder of the input/output cycle. Accordingly, AD10 goes low, AD11 goes high, and, at write strobe (NWDS) time, the conversion starts. Since there are seven instructions executed prior to the Load Instruction, the NOPS are not required unless the converter clock rate is slowed down. If the clock rate is considerably less than 1.0 megahertz, more delay may be required – refer to appendix E for delay calculations.

Once the analog-to-digital conversion is completed (after 40 clock cycles), the digital data are accepted by SC/MP and are stored in a specified memory location. As indicated by lines 15 through 20 of the program, P2 is loaded with the address for "data acceptance" – in this case, X'0400 – and P3 is loaded with the "memory-destination" address (X'0200). When the LD Instruction (line 21) is executed, the 8-bit digital output of the converter is read into the Accumulator, and when the next instruction (STore, line 22) is executed, the data are stored in memory location X'0200. Observe that when the X'0400 address is valid, the output data are gated into the Accumulator via the address strobe (NADS) rather than the read strobe (NRDS); the NADS signal provides adequate time for the transfer of data, whereas the NRDS signal may not.

ANALOG-TO-DIGITAL CONVERSION USING MULTIPLE CONVERTERS

General Description

The SC/MP-based multiple analog-to-digital converter shown in figure 2C1-5 is simply an extension of the principles and concepts used in the single configuration (figure 2C1-2). The multiple system is adaptable to almost any application where analog-to-digital conversion is required – complex control systems, multiple-parameter testing, precision measurements, environmental studies, and many others. Among other advantages, the multiple-converter system is easy and relatively inexpensive to implement.

System Operation

Other than supply voltages, the multiple system requires start/select signals for each converter, a user-supplied clock source, output buffers, and appropriate control and decoding logic.

NOTE

In figure 2C1-5, it is assumed that each analog signal source is located some distance from the other; thus, the analog-to-digital conversion is performed at the origin rather than at the destination. This is done because a digital signal can be transmitted over a reasonable distance with little or no degradation, whereas an analog signal may require additional circuits for equivalent accuracy and stability. If the signal sources are in close proximity and sampling time is not a critical factor, the inputs can be multiplexed as shown in figure 2C1-7.

Some of the multiple-converter circuits are functionally equivalent to those of the single converter; thus, they are described by reference only.

The eight analog-to-digital converters shown in figure 2C1-5 can be started and the associated output selected in any sequential order – this being determined by the users program and being implemented by the two BCD-to-decimal decoders. The 'start' and 'select' decoders operate as follows. A starting address of X'0800 has been arbitrarily selected; any other nonconflicting address can be used. The chosen address is recognized when address bit 11 (AD11) is high and address bit 10 (AD10) is low. When the write data strobe (NWDS) goes active-low, the decoder input logic (G1, G2A, and G2B) is enabled and address bits 0, 1, and 2 are decoded to produce any one of eight start signals. The start decoding truth table is shown as an inset to the upper BCD-to-decimal decoder block of figure 2C1-5.

The 'select' decoder uses an arbitrary address of X'0400 and is synchronized by the address strobe (NADS) from SC/MP. With address bit 10 (AD10) high and address bit 11 (AD11) low, a logic '1' appears at pin 2 of the flip-flop. At NADS time, a high also appears at pin 3 of the flip-flop; accordingly, the $\overline{Q}$ output (pin 6) is driven low. Likewise, the 'D' input (pin 12) of the decoder is low and address bits 0, 1, and 2 are decoded to produce any one of eight 'select' signals. The converter-select signal remains valid until the bus request line is released by the microprocessor (BREQ goes low) – at which time, the decoding logic is disabled ($\overline{Q}$ goes high).

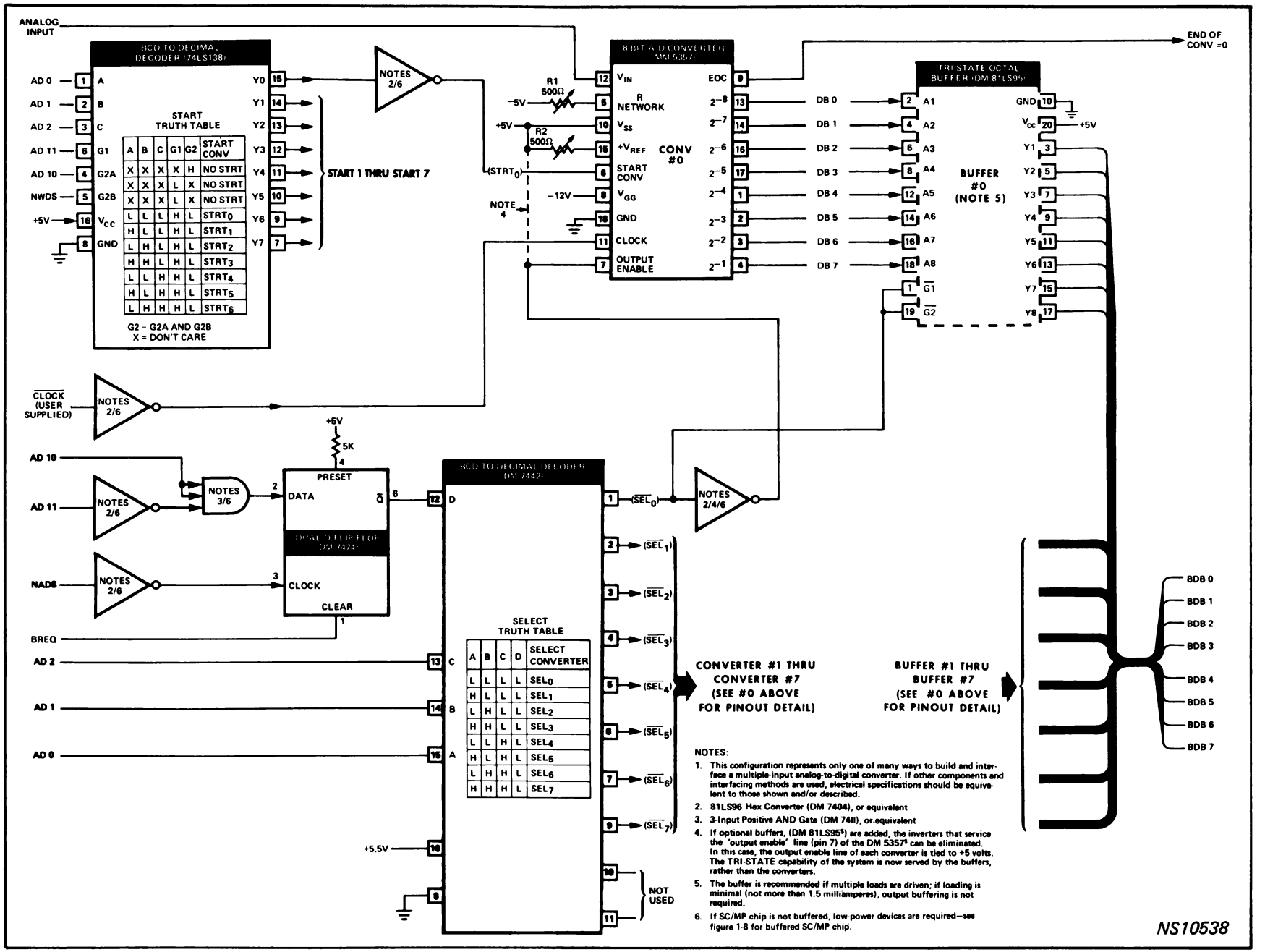


Figure 2C1–5. Multiple-Converter Analog-to-Digital System

In a multiple-converter system, timing for one conversion cycle is identical to that shown in figure 2C1-3; however, there are noteworthy differences in the start and select functions. The eight converters in figure 2C1-5 can be started in any random order and, after an appropriate delay (40 x clock period), the outputs can be selected at random. For instance, it may be desirable to start the converters in 2-6-1-7-4-3-8-5 order and to enable the outputs in 7-2-5-4-1-3-8-6 order, or in some other sequence that suits a particular application. Output levels (DB0 through DB7) of each converter are TTL-compatible and, as previously indicated, the data lines are TRI-STATE to permit common bus lines. If the output lines must drive multiple loads, a TRI-STATE buffer such as the DM81LS95 (shown as a broken-line block in figure 2C1-5), is required.

System Adjustments

The zero (R1) and full-scale (R2) adjustments for each converter in figure 2C1-5 are identical to those described for the single digital-to-analog system (figure 2C1-2).

Software Considerations

The flowchart and the program listing in figure 2C1-6 shows how the multiple analog-to-digital converter system (figure 2C1-5) can be software-controlled to provide the functions described under system operation. Software concepts are identical to those shown and described for the single-converter system except that eight converters must be serviced rather than just one converter.

An address of X'080N – where N = 0 through 7 – is selected to start the conversion; line 10 of the multiple-converter program (figure 2C1-6) stores to the address of the first converter (X'0800). When the program is executed by SC/MP, address bits 0, 1, and 2 (figure 2C1-5) are modified by lines 11 through 17 of the program and are decoded by the 74LS138 chip; in this case, the converters are started in a 1-through-8 sequence. Any other starting sequence can be obtained by rearrangement of the store instructions. The last Store Instruction is followed by four NOP Instructions, each consuming 10 microseconds; thus, a delay of 40 microseconds is introduced. Assuming a 1.0-megahertz clock, no delay is required for this program; the NOP instructions are included only to indicate how a 'short' delay can be implemented. If a 100-kilohertz clock is used, the conversion time is 400 microseconds (that is, 40 x 1/fclock). In this case, a delay of not less than 104 microseconds is required – assuming the converter outputs are loaded and stored in the "starting" sequence. If the load/store operations are performed in random order, the worst case delay is 292 microseconds – refer to appendix E for program delays of this magnitude.

After the analog-to-digital conversions are complete and select address X'040N – where N = 0 through 7 – is valid, the digital outputs of converters 1 through 8 are loaded into the Accumulator and are transferred to the memory location pointed to by P3. In likeness to the single-converter system, the output data are gated into the Accumulator via the address strobe (NADS); the slow access time of the converter peripherals does not permit the use of the read strobe (NRDS). As previously indicated, the program can be arranged to allow sampling of the converter outputs in any random order.

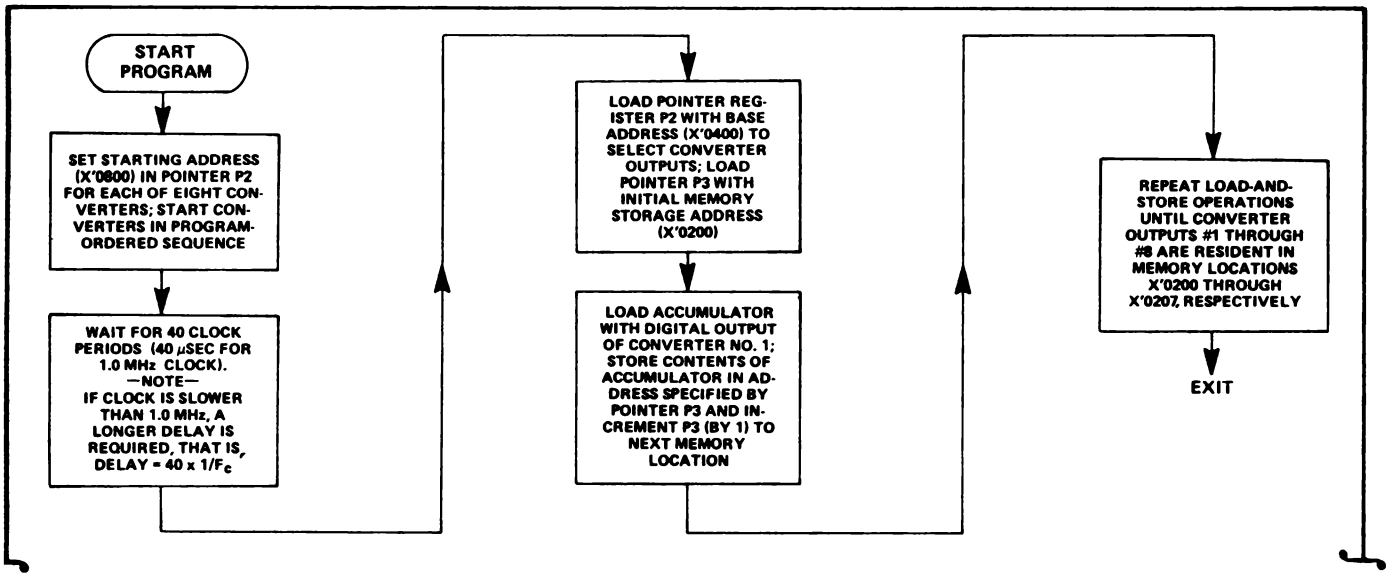


Figure 2C1–6. Flowchart and Program Listing for Multiple-Converter System

```
 1                          TITLE  SCMP, 'MULTIPLE A/D CONVERTER PRGM'
 2
 3        0002  P2        =       2
 4        0003  P3        =       3
 5
 6  0000  08              NOP
 7  0001  C400  START:    LDI     0           ;BASE ST ADRS -> PTR2
 8  0003  32              XPAL    P2
 9  0004  C408            LDI     8
10  0006  36              XPAH    P2
11  0007  CA00            ST      CONV0(P2)   ;START CONVERTERS 0 - 7
12  0009  CA01            ST      CONV1(P2)
13  000B  CA02            ST      CONV2(P2)
14  000D  CA03            ST      CONV3(P2)
15  000F  CA04            ST      CONV4(P2)
16  0011  CA05            ST      CONV5(P2)
17  0013  CA06            ST      CONV6(P2)
18  0015  CA07            ST      CONV7(P2)
19  0017  08              NOP                 ;IF REQUIRED, DELAY
20  0018  08              NOP                 ;INTRODUCED TO COMPLETE
21  0019  08              NOP                 ;THE CONVERSION.
22  001A  08              NOP
23  001B  C404  ACCEPT:   LDI     4           ;LD P2 WITH BASE CONVRTR ADRS
24  001D  36              XPAH    P2
25  001E  C402            LDI     2           ;LD P3 WITH MEM ADRS FOR
26  0020  37              XPAH    P3          ;CONVERTED DIGITAL OUTPUT
27  0021  C400            LDI     0
28  0023  33              XPAL    P3
29  0024  C200            LD      CONV0(P2)
30  0026  CF01            ST      @1(P3)
31  0028  C201            LD      CONV1(P2)
32  002A  CF01            ST      @1(P3)
33  002C  C202            LD      CONV2(P2)
34  002E  CF01            ST      @1(P3)
35  0030  C203            LD      CONV3(P2)
36  0032  CF01            ST      @1(P3)
37  0034  C204            LD      CONV4(P2)
38  0036  CF01            ST      @1(P3)
39  0038  C205            LD      CONV5(P2)
40  003A  CF01            ST      @1(P3)
41  003C  C206            LD      CONV6(P2)
42  003E  CF01            ST      @1(P3)
43  0040  C207            LD      CONV7(P2)
44  0042  CF01            ST      @1(P3)
45
46                EXIT:                       ;USER RETURN ROUTINE
47
48        0000  CONV0     =       0
49        0001  CONV1     =       1
50        0002  CONV2     =       2
51        0003  CONV3     =       3
52        0004  CONV4     =       4
53        0005  CONV5     =       5
54        0006  CONV6     =       6
```

Figure 2C1–6 (Continued)

```
55      0007  CONV7   =       7
56
57      0001            END     START


ACCEPT  001B *      CONV0   0000        CONV1   0001
CONV2   0002        CONV3   0003        CONV4   0004
CONV5   0005        CONV6   0006        CONV7   0007
EXIT    0044 *      P2      0002        P3      0003
START   0001



NO ERROR LINES
SOURCE CHECKSUM=0C47
```

NS10539

Figure 2C1–6 (Concluded)

ANALOG-TO-DIGITAL CONVERTER USING MULTIPLEXED INPUTS

General Description

Except for the input multiplexer, the analog-to-digital converter system shown in figure 2C1-7 is functionally equivalent to the multiple-converter system (figure 2C1-5). Both systems can be readily adapted to such applications as complex controllers, multiple-parameter testing, precision measurements, and numerous other applications that fall within the analog-to-digital and digital-to-analog class. The multiplexed-input system is particularly well-suited to applications where a multiple-input profile is required and where cost is an absolute restraining factor. Assuming reasonable input sources and sound transmission techniques, no input calibration is required and, as shown in figure 2C-7, the saving of hardware is appreciable. The multiplexed-input system can be expanded easily to serve most output requirements – refer to note 4 of figure 2C1-7 for one possible technique.

System Operation

The 'start' and 'select' procedures for the multiplexed-input system are identical to those described for the multiple system shown in figure 2C1-5 – a starting address of X'080N is arbitrarily selected and the output is enabled at address X'040N; in both cases, 'N' = 0 through 7. The start function is implemented at write strobe (NWDS) time, whereas the output is enabled at address strobe (NADS) time; the output-enable signal remains valid until the output bus is released by the microprocessor (BREQ goes low).

The analog input (V_{IN}/pin 12) to the converter is selected via an 8-channel multiplexer (AM3705), which is driven by a clocked latch. A quad-D flip-flop (DM8551) is used to perform the latching function; the input disable gates (pins 9/10) eliminate the possibility of false clocking and, at the same time, ensure that valid output codes are maintained during the clock period. At read strobe (NRDS) time, outputs A, B, and C of the latch are set to agree with the binary pattern of address bits 0, 1, and 2. These three outputs are then decoded in the multiplexer to select one of eight analog inputs; the selected signal is outputted at pin 3 and serves as the input for the analog-to-digital converter. The signal select decoding truth table is shown as an inset to the multiplexer block in figure 2C1-7.

Under program control, analog input signals #1 through #8 can be selected in any random order; in fact, if the application requires, the same signal can be examined over and over again. When a particular conversion is complete, the end-of-conversion (EOC) gate goes high and the 8-bit digital word is loaded into an on-chip latch. The output data lines (DB0 through DB7) are TRI-STATE to permit the use of common system buses and are TTL-compatible for fanout flexibility.

System Adjustments

The zero (R1) and full-scale (R2) adjustments for the converter are identical to those described for the single-input analog-to-digital system (figure 2C1-2).

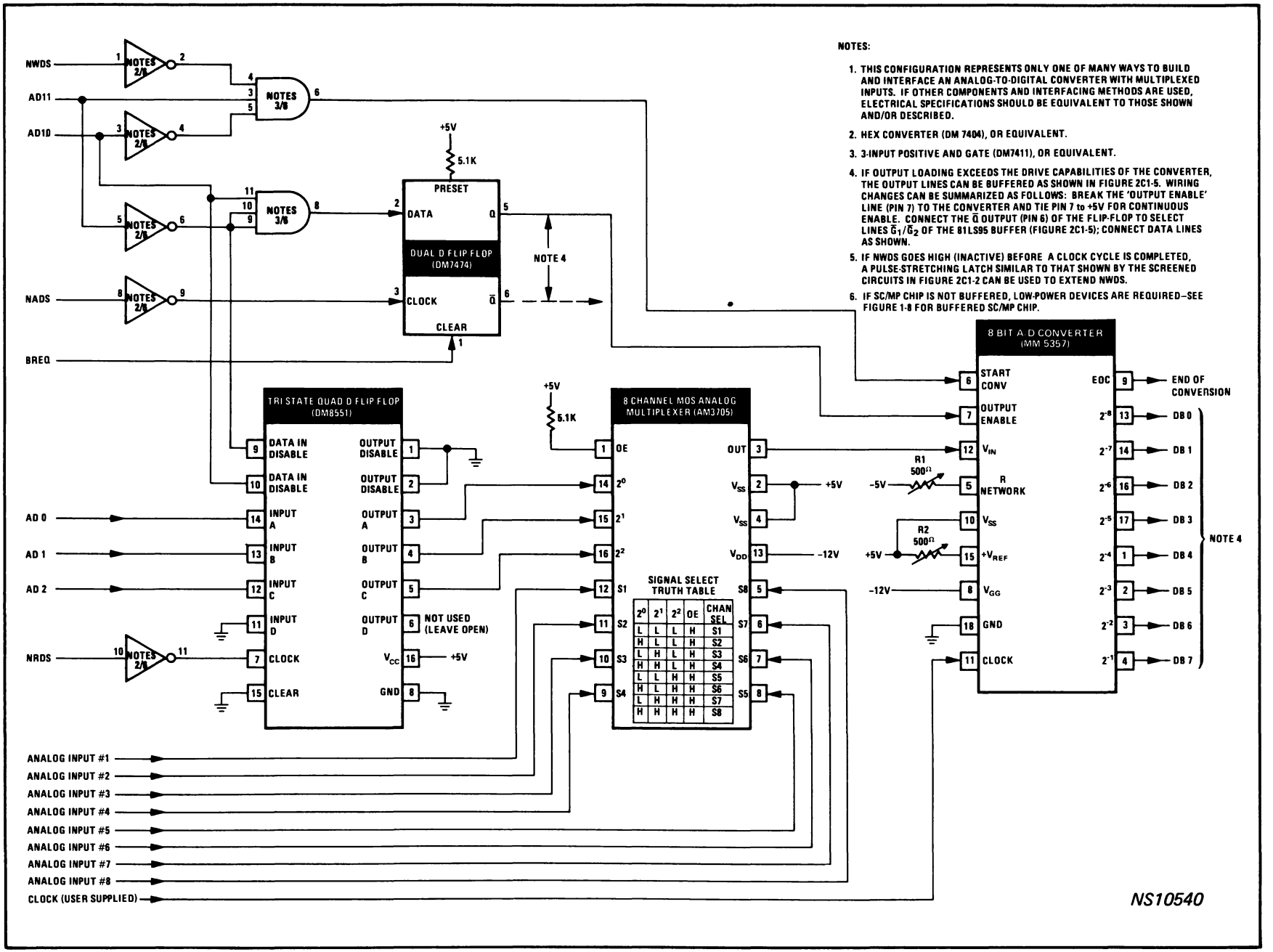
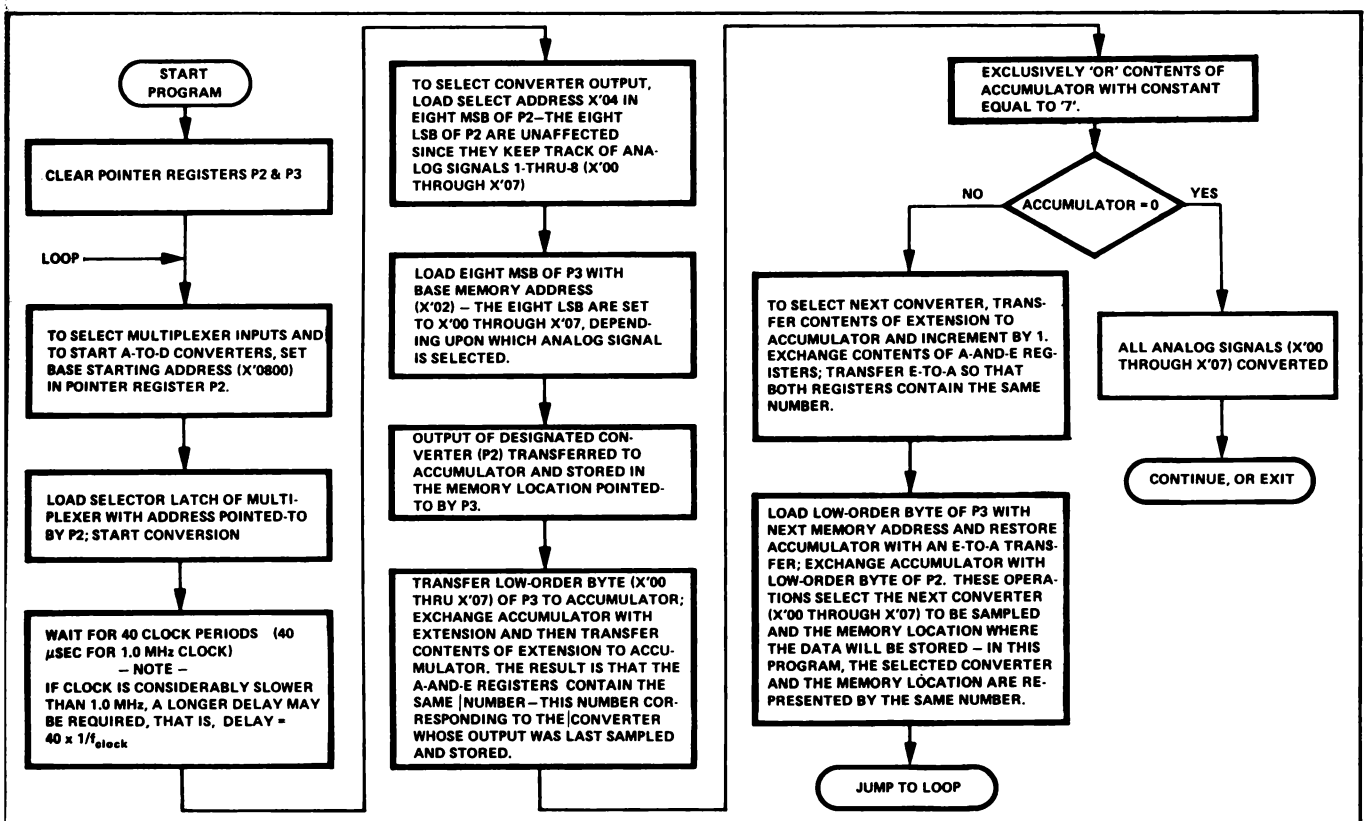


Figure 2C1–7. Analog-to-Digital Converter System Using Multiplexed Inputs

Software Considerations

Figure 2C1-8 shows the flowchart and the program listing that are applicable to the multiplexed-input system of figure 2C1-7. The basic concepts are the same as those of the single- and multiple-converter systems – that is, start the converter, accept the 8-bit digital data, and store the output in memory.

The program is set up to start the converters in a 1-to-8 sequence (hexadecimal address X'00 through X'07); this is an arbitrary choice and can be altered to select any one of the converters by changing the 8 low-order bits of P2. Since there is a one-to-one correspondence between the selected converter and the memory storage location, converter X'0800 is stored in memory location X'0200, converter X'0801 in location X'0201, and so on for the remaining six converters. The memory addresses likewise should be altered by appropriate changes in the 8 low-order bits of P3. After each conversion, the program checks to see if all eight analog-to-digital operations are completed; as shown by the flowchart, this is done by inclusively ORing the Accumulator with a constant – in this case, '7'. If the OR-result is not equal to '0', the program loops back and continues with the next conversion. If the OR-result is '0', the eighth conversion is completed and, as written, the program will exit to an appropriate users routine.




```
 1                      TITLE  SCMP, 'MULTIPLE A/D CNVTR WITH ANALOG MUX'
 2        0001  P1       =      1
 3        0002  P2       =      2
 4        0003  P3       =      3
 5
 6  0000  08             NOP
 7  0001  C400  START:   LDI    0        ;BASE STARTING ADRS
 8  0003  32             XPAL   P2
 9  0004  C400           LDI    0
10  0006  33             XPAL   P3
11  0007  C408  LOOP:    LDI    08       ;GENERATE ANALOG SWITCH ADRS
```

Figure 2C1–8. Flowchart and Program Listing for Analog-to-Digital Converter System with Multiplexed Inputs

```
12 0009 36              XPAH    P2
13 000A C200            LD      (P2)    ;LOAD ANALOG SWITCH ADRS LATCH
14 000C CA00            ST      (P2)    ;START CONVERSION
15 000E C404            LDI     4
16 0010 36              XPAH    P2
17 0011 C402            LDI     2
18 0013 37              XPAH    P3      ;CONVERSION COMPLETE IN 40 MIC
19 0014 C200            LD      (P2)    ;ACCEPT CONVERTER OUTPUT
20 0016 CB00            ST      (P3)    ;STORE CONVERTER OUTPUT
21 0018 33              XPAL    P3
22 0019 01              XAE
23 001A 40              LDE
24 001B E407            XRI     7       ;CHECK IF ALL ANALOG
25 001D 980A            JZ      EXIT    ;SIGNALS CONVERTED
26 001F 40              LDE
27 0020 F401            ADI     1       ;ADRS FOR NEXT ANALOG SIGNAL
28 0022 01              XAE
29 0023 40              LDE
30 0024 33              XPAL    P3
31 0025 40              LDE
32 0026 32              XPAL    P2
33 0027 90DE            JMP     LOOP
34
35              EXIT:                   ;USER RETURN ROUTINE
36
37      0000              END

EXIT    0029        LOOP    0007        P1      0001 *
P2      0002        P3      0003        START   0001 *
```

NS10541

Figure 2C1–8 (Concluded)

CONCEPTS FOR A LOW-COST SYSTEM

General Description

The 4-chip system shown in figure 2C1-9 illustrates a very simple technique for converting a digital input to an analog output and also demonstrates how this analog signal can be used to generate the digital equivalent of some unknown voltage. Converter systems of this type can be usefully employed in such applications as security/alarm systems, error-control loops, digital plotters, and any other application where these techniques are applicable. The connection scheme shown in figure 2C1-9 requires a dedicated microprocessor during the conversion cycle and the scheme is limited to a single 8-bit output. However, this low-cost system requires little external hardware, no bus connections, and only a few control signals. The system is further enhanced by the fact that Sense B, the serial input/output capability, and flags 0, 1, and 2 are not used, so conceivably these resources of SC/MP can be put to use in other applications.

System Operation

As shown, the 8-bit data word (DB0 through DB7) enters two flip-flops that serve as input latches. When a valid address is received – hexadecimal address X'0400 has been arbitrarily chosen – the 'data input disable' lines (pins 9 and 10) are driven low. At write strobe (NWDS) time with both disable lines low, outputs D, C, B, and A of one latch are set to agree with the logic states of the 4 least significant bits (DB0, 1, 2, and 3, respectively), and the other latch is set to agree with the logic states of the 4 most significant bits. Thus, the 8-bit word is latched and is applied to the DA1200 for the digital-to-analog conversion process. The digital-to-analog converter uses a series of current-weighted switches, an ultrastable resistor network, a precision voltage reference, and three high-gain operational amplifiers to produce the analog-voltage equivalent of the 8-bit digital input.

The input currents are summed and compared to a precision reference voltage; in the system shown, the +10-volt reference (pin 14) is supplied externally – although the internal

reference can be used for most purposes. The difference voltage that results from comparing the weighted input currents (pin 19) of the digital word. It is readily seen that if the digital input consists of all ones (FF), the analog output is 10 volts; whereas, if the input word consists of all zeroes, the analog output is 0 volt. Suppose the input bit pattern is 10000000 (80); since this number is halfway between 00 and FF, the analog output is very close to 5 volts. This analog output can be buffered and used as an error-control signal or for any other purpose that the application requires.

The system shown in figure 2C1-9 can also be used to produce an 8-bit word that is the digital equivalent of an unknown voltage. Such an unknown voltage – within the conversion range – is shown connected to pin 3 of the voltage comparator (LM311); pin 2 of the comparator is connected to the output of the digital-to-analog converter. Using successive approximation techniques, the inputs at pins 2 and 3 are compared and, under software direction, appropriate adjustment is made to the input word (DB0 through DB7) after each approximation. With some assumptions, the operation can be summarized as follows. Since the inherent resolving power of the system is 8 bits, there are 256 digital increments from one end of the conversion range to the other; that is, if 0 volt is represented by a bit pattern of all zeroes, the 10-volt limit is represented by a bit pattern of all ones. Assuming an unknown value of 7 volts, the input word for the first approximation is set to the halfway point – in hexadecimal format X'80. Since the unknown voltage is higher than the analog-voltage equivalent of the first input approximation, the Sense-A line of SC/MP is driven high. The software differentiates between these two conditions (Sense A = '0' and Sense A = '1') in such a way that if the line is high, the input word is set halfway between X'80 and X'FF. If the sense line is low, the input word is set halfway between X'00 and X'80. In the current example, the input word is set to X'C0 and a second approximation is performed. Assuming a digital-to-analog relationship that is nearly linear, the output at pin 19 (approximately 7.5 volts) is now higher than the unknown and the Sense-A line is driven low. A third approximation is made between X'C0 X'80, that is, at X'A0. Similar input adjustments based on the high or the low state of sense A are contained until eight approximations are completed; at this time, an 8-bit word that is very near the analog equivalent of the unknown voltage will appear in the Accumulator of SC/MP. This word can be stored away and used later to plot the unknown voltage or for any other purpose the application requires.

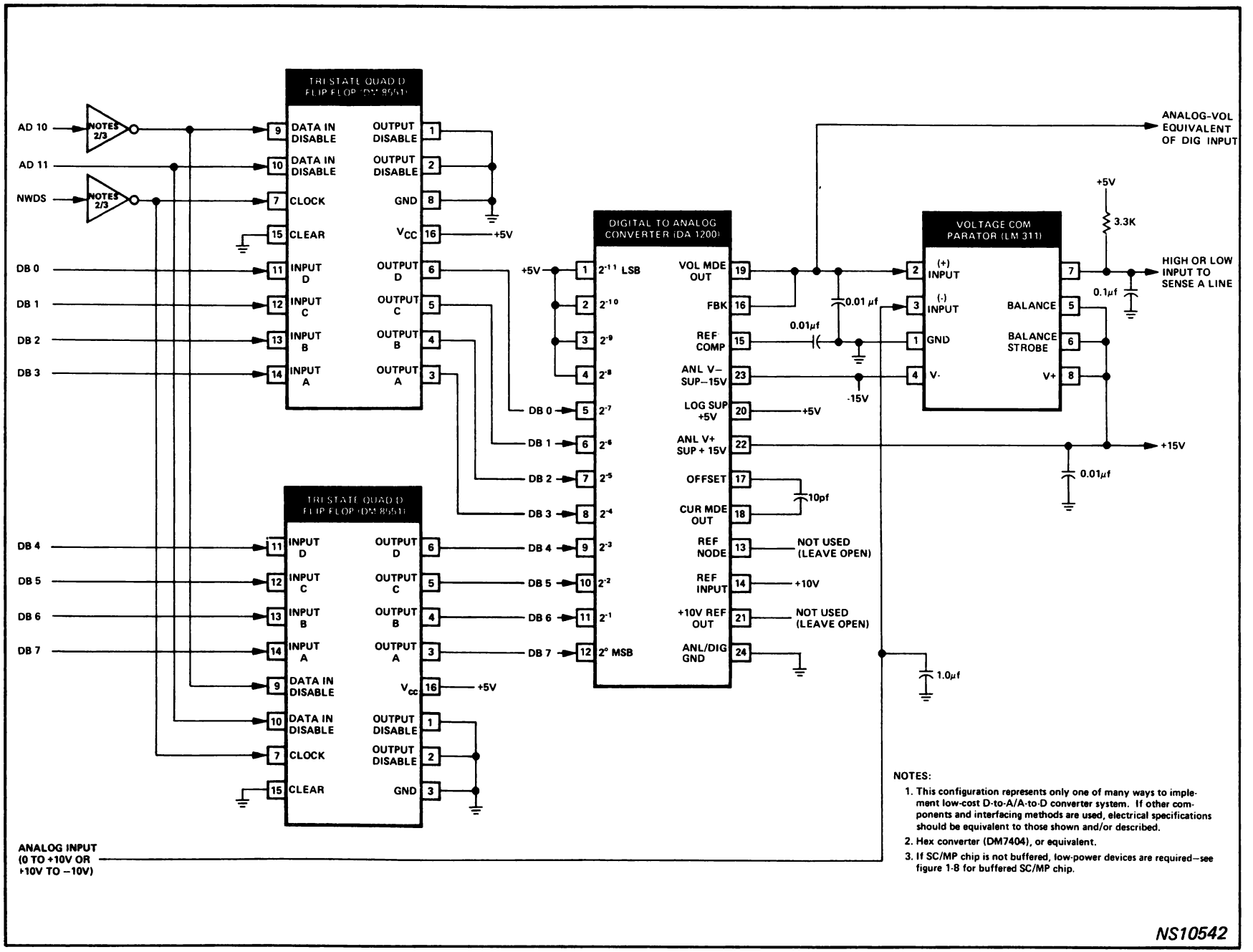
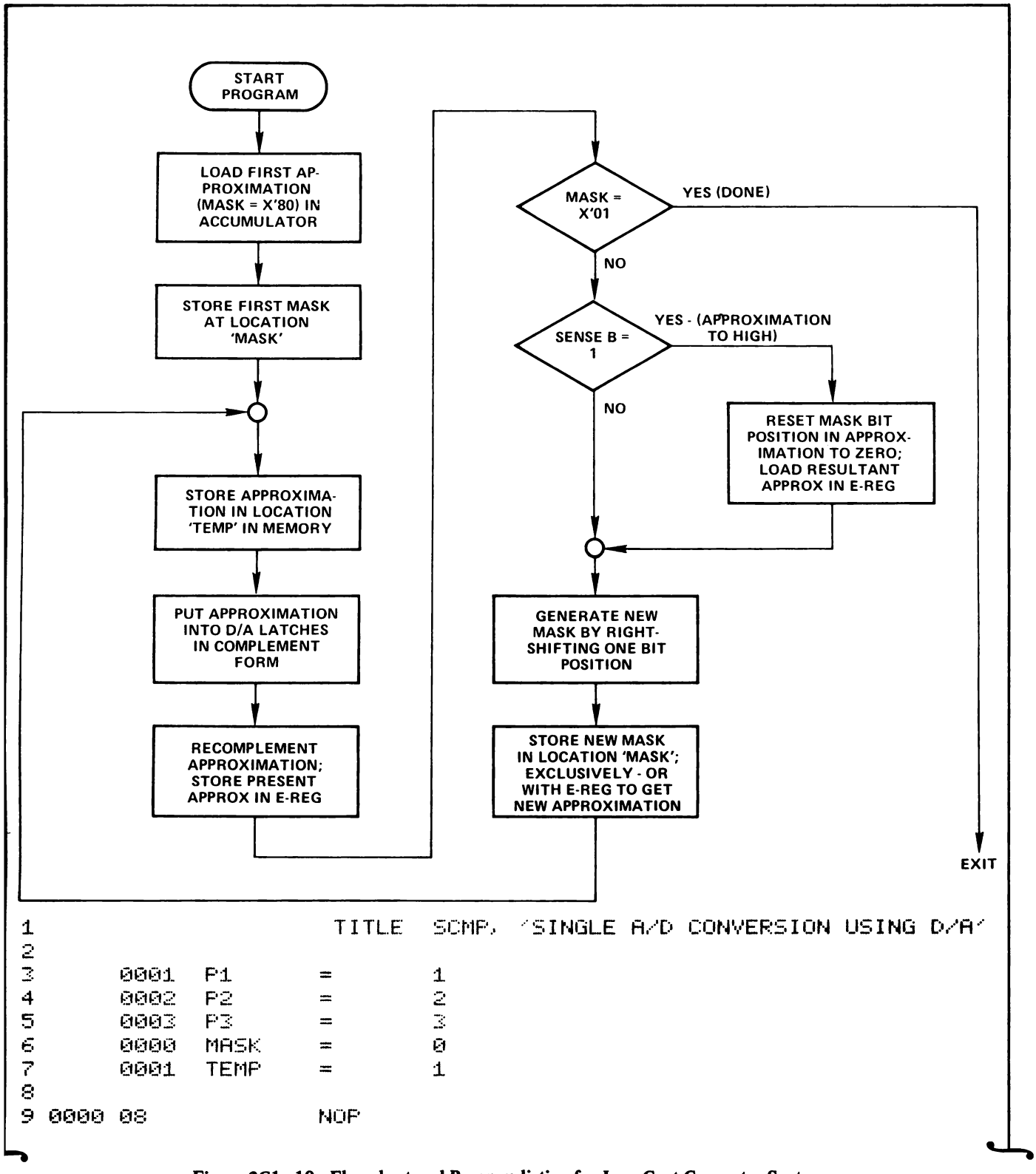


Figure 2C1–9. Low-Cost Converter System

Software Considerations. The flowchart and program listing shown in figure 2C1-10 shows one method of implementing software control of the low-cost D-A/A-D system. The software can easily be expanded to utilize the analog data for whatever purpose the application requires.




```
1                       TITLE  SCMP, 'SINGLE A/D CONVERSION USING D/A'
2
3       0001  P1        =      1
4       0002  P2        =      2
5       0003  P3        =      3
6       0000  MASK      =      0
7       0001  TEMP      =      1
8
9 0000  08              NOP
```

Figure 2C1–10. Flowchart and Program listing for Low-Cost Converter System

```
10 0001 C400  CONV:   LDI     0          ;SETUP PTR REG ADRS
11 0003 32            XPAL    P2
12 0004 C400          LDI     0
13 0006 33            XPAL    P3
14 0007 C402          LDI     2
15 0009 37            XPAH    P3
16 000A C408          LDI     8
17 000C 36            XPAH    P2         ;ADRS OF D/A CONVRTR IN P2
18 000D C480  NEXT:   LDI     080        ;FIRST APPROX TO ACU
19 000F CB00          ST      MASK(P3)   ;SAVE APPROX TO USE AS
20                                       ;A MASK
21 0011 CB01  AGAIN:  ST      TEMP(P3)   ;STORE APPROX IN TEMP
22 0013 C400          LDI     0          ;COMPLEMENT INPUT
23 0015 FB01          CAD     TEMP(P3)
24 0017 CA00          ST      0(P2)      ;APPROX TO D/A LATCH
25 0019 CB01          ST      TEMP(P3)
26 001B C400          LDI     0
27 001D FB01          CAD     TEMP(P3)   ;RECOMPLEMENT APPROX
28 001F 01            XAE                ;APPROX TO EXT REG
29 0020 C300          LD      MASK(P3)   ;MASK TO ACU
30 0022 E401          XRI     1          ;COMPARE MASK TO X'01
31 0024 9813          JZ      EXIT       ;APPROX COMPLETE
32 0026 06            CSA                ;STATUS TO ACU
33 0027 D420          ANI     020        ;CHECK SENSE B
34 0029 9C08          JNZ     HIGH       ;SENSE B HIGH
35 002B C300  SHIFT:  LD      MASK(P3)   ;MASK TO ACU
36 002D 1C            SR                 ;SHIFT RIGHT MASK
37 002E CB00          ST      MASK(P3)   ;STORE NEXT APPROX
38 0030 60            XRE                ;SET NEXT LOW ORDER
39                                       ;BIT IN EXT REG
40 0031 90DE          JMP     AGAIN
41 0033 C300  HIGH:   LD      MASK(P3)   ;MASK TO ACU
42 0035 60            XRE                ;RESET BIT TO 0
43 0036 01            XAE                ;APPROX TO EXT REG
44 0037 90F2          JMP     SHIFT      ;JMP TO SHIFT MASK
45                                       ;AND SET NEXT BIT
46            EXIT:                      ;EXIT WITH THE VALUE OF ANALOG
47                                       ;SGNAL IN EXT REG.
48
49      0000           END

AGAIN   0011         CONV    0001 *       EXIT    0039
HIGH    0033         MASK    0000         NEXT    000D *
P1      0001 *       P2      0002         P3      0003
SHIFT   002B         TEMP    0001


NO ERROR LINES
SOURCE CHECKSUM=4F0F
```

NS10543

Figure 2C1–10 (Concluded)

INTERFACING A KEYBOARD TO SC/MP

SC/MP applications that require a keyboard interface usually use one of two methods to generate keycodes. In one method, SC/MP is used as a keyboard scanner, whereas in the other method, SC/MP is interfaced with a keyboard encoder. For either type of interface, programs can be developed for continuous keyboard scan or for using the keyboard as an interrupt device.

When SC/MP is interfaced with an appropriate keyboard and is supported with the proper software, any application that can be controlled by alphanumeric inputs is feasible – lawn-sprinkler control, home and business lighting, vending machines, combination locks, kitchen appliances, games, and so on. Some basic principles of keyboard interfacing are shown in the following illustrations and are described in the supporting text.

USING SC/MP AS A KEYBOARD SCANNER

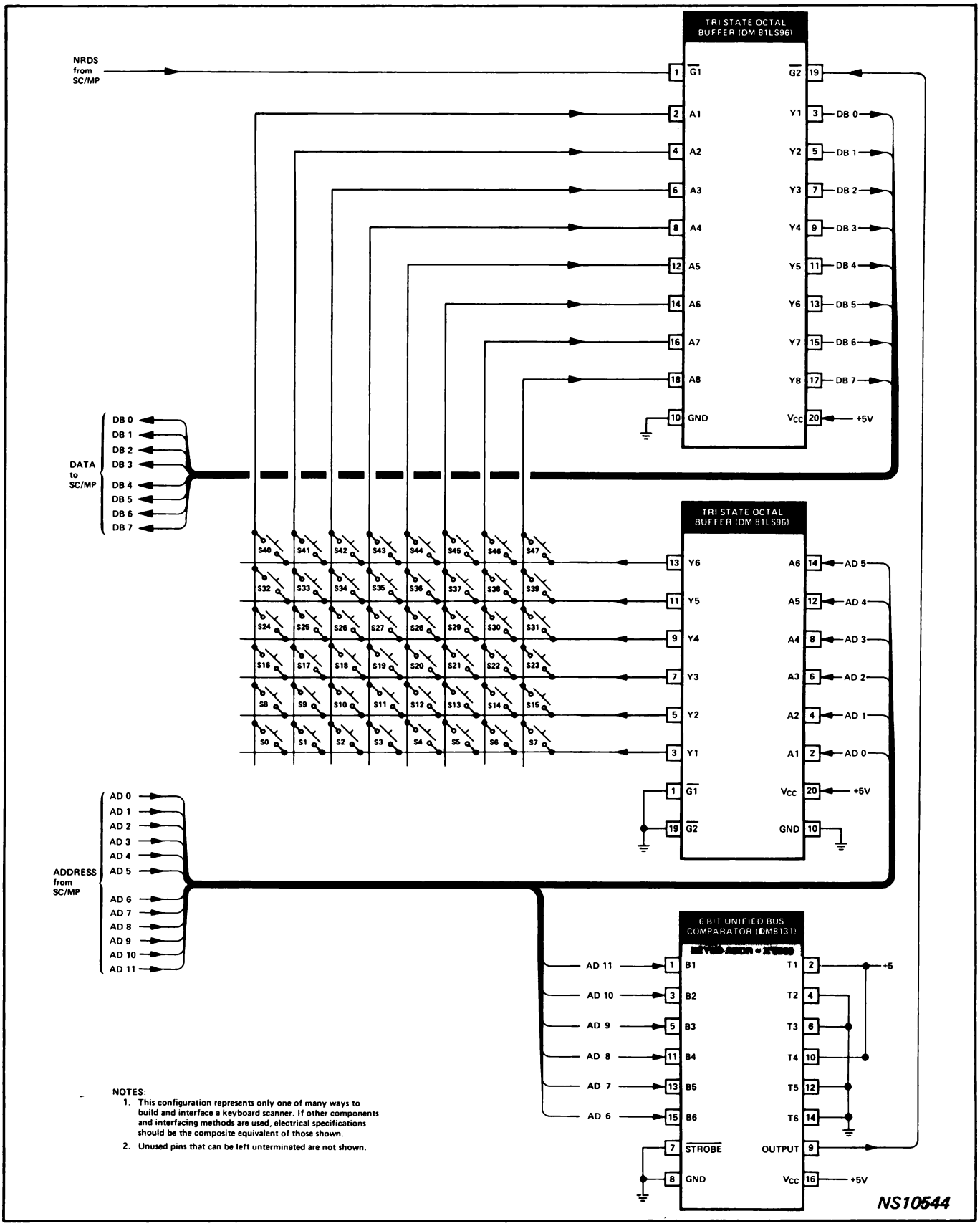
General Description

The keyboard matrix shown in figure 2C2-1 consists of six rows with eight keys in each row. Functional relationships between the keyboard and SC/MP can be summarized as follows. The entire key matrix is scanned by testing input data to the microprocessor for a value other than zero; this condition occurs if any key is depressed. After key detect, a software debounce is performed. Then, the program determines the row and the column corresponding to the key, computes the correct binary code, tests for key release, puts the keycode in the SC/MP Extension Register, and returns to the calling program.

System Operation

A fixed address (X'0900) is assigned to the "keyboard peripheral" and when a load (LD) instruction is executed, the TRI-STATE buffers are simultaneously activated. Thus, if any key (S0 through S47) is pressed, one of the bits (DB0 through DB7) on the data bus appears as a logic '1'.

The program "LOOP" checks for the "nonzero" condition and provides a debounce time of 5 milliseconds. After debounce, the value of the key is determined by updating a counter in RAM. The counter is incremented by "8" for each row scanned and by "1" for each column scanned. For example, assume that S9 (row 2/column 2 of figure 2C2-1) is pressed. The first row is scanned by the software, and finding no key pressed, the row counter (row select) is incremented by 8 and the second row is scanned. In this row, one of the bits in the data word is a logic '1'; accordingly, the column counter is now incremented by 1 and comparisons are made to determine which switch is pressed – for this example, it is the second switch (S9) in the column. It can readily be seen that a different binary code (keycode) is produced for each switch in the matrix. The keycode is saved in temporary memory, and the keyboard is tested for key release by executing a Load Instruction to the keyboard. This activates all the buffers, and, upon key release, the keycode is transferred from temporary memory to the Extension Register. The designated pointer then is exchanged with the Program Counter to return to the calling program.



NOTES:

1. This configuration represents only one of many ways to build and interface a keyboard scanner. If other components and interfacing methods are used, electrical specifications should be the composite equivalent of those shown.
2. Unused pins that can be left unterminated are not shown.

Figure 2C2–1. Using SC/MP as A Keyboard Scanner

The flowchart in figure 2C2-2 and the program listing in figure 2C2-3 show how SC/MP can be utilized by software to perform a keyboard scanning function.

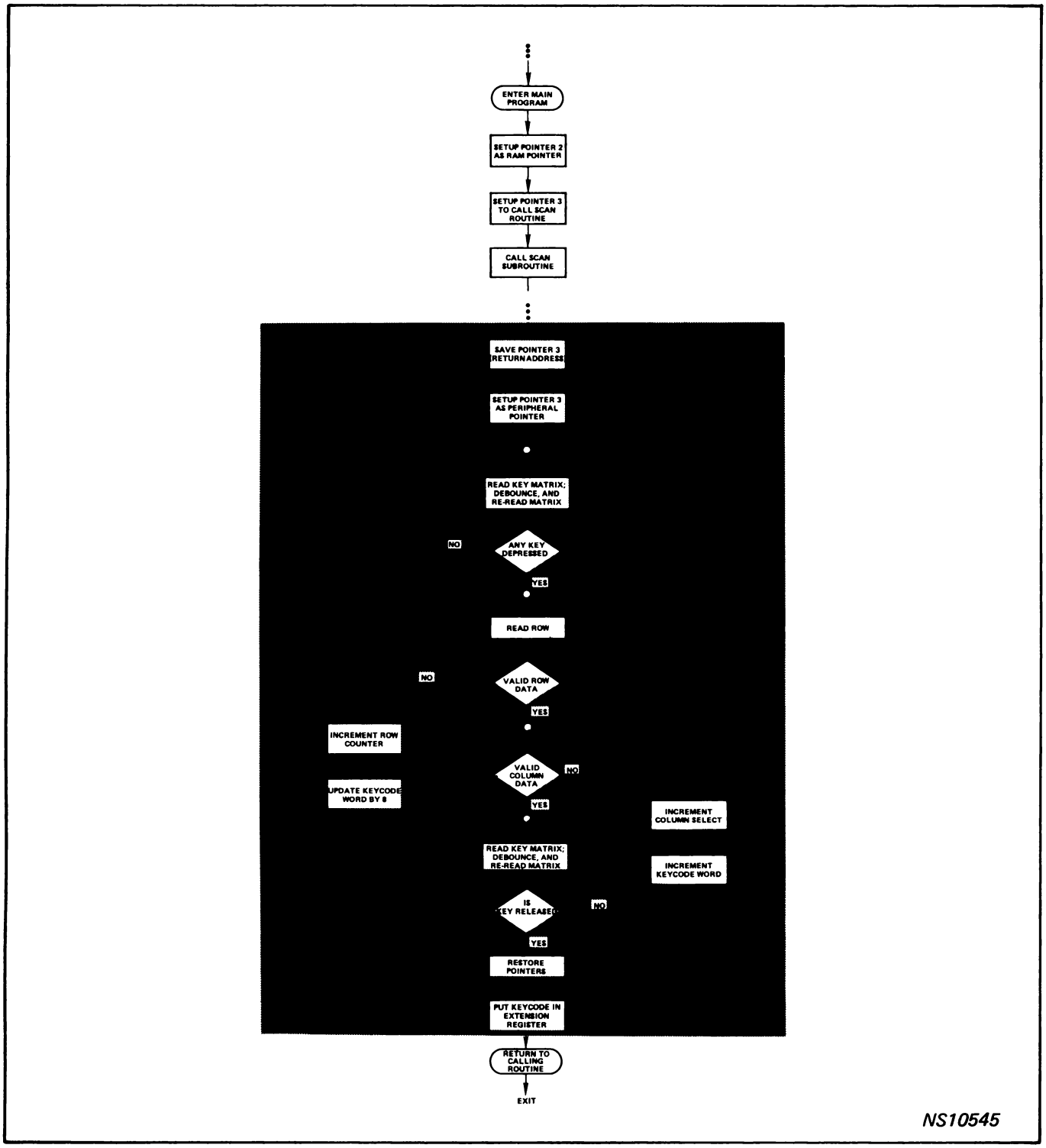


Figure 2C2–2. Flowchart for SC/MP Interfaced with A 6×8 Keyboard Matrix

```
                        TITLE SCANNR, ' SC/MP'

              ;SC/MP ACTS AS A KEYBOARD SCANNER.  THE KEYBOARD
               IS A 6 BY 8 MATRIX.




                   ;TYPICAL MAIN PROGRAM


              ;THIS SECTION OF CODE SETS UP THE RAM POINTER
              ;AND THE POINTER FOR THE SCAN ROUTINE.


0000 C400              LDI   L(RAM)
0002 32               XPAL   2
0003 C403              LDI   H(RAM)
0005 36               XPAH   2
0006 C40F              LDI   L(SCAN)-1
0008 33               XPAL   3
0009 C400              LDI   H(SCAN)
000B 37               XPAH   3
000C 3F               XPPC   3            ;CALL SCAN ROUTINE
000D 40     START:     LDE                ;PUT CODE INTO ACCUM
000E CA00               ST   SAVE(2)       ;SAVE CODE IN RAM


                ;USERS PROGRAM WOULD BEGIN HERE AND OPERATES
                ;ON THE KEY CODE STORED IN LOCATION 'SAVE'









              ;THE SCAN ROUTINE FORMS THE CORRECT CODE IN
              ;LOCATION 'SWITCH'   AFTER CHECKING FOR KEY
              ;RELEASE.  CONTROL IS RETURNED TO THE CALLING
              ;PROGRAM WITH THE KEYCODE IN THE E REGISTER.


0010 33     SCAN:      XPAL 3
0011 CA01               ST   TEMPL(2)      ;SAVE PTR 3 LO
0013 37               XPAH   3
0014 CA02               ST   TEMPH(2)      ;SAVE PTR 3 HI
0016 C400              LDI   L(PERIPH)
0018 33               XPAL   3
0019 C409              LDI   H(PERIPH)
```

Figure 2C2–3. Program Listing for SC/MP Interfaced with A 6×8 Keyboard Matrix

```
 55 001B 37                XPAH  3              ;SET UP PERIPHERAL
 56                                             ;POINTER
 57 001C 02     OVER:       CCL
 58 001D C400               LDI   0
 59 001F CA03                ST   SWITCH(2)     ;CLR SWITCH WORD
 60
 61
 62
 63                       ;THE ACTUAL SCANNING OF THE KEYBOARD
 64                       ;FORMING THE KEYCODE AND TESTING FOR
 65                       ;KEY RELEASE BEGINS WITH THE LABEL 'LOOP'
 66                       ;AND ENDS WITH THE INSTRUCTION "JNZ RELEAS"
 67
 68
 69 0021 C33F   LOOP:        LD   ALLKYS(3)
 70 0023 01                 XAE                ;SAVE CODE
 71 0024 8F05               DLY   5            ;DEBOUNCE 5 MS
 72 0026 C33F                LD   ALLKYS(3)
 73 0028 50                 ANE                ;COMPARE NEW WITH OLD
 74 0029 98F6                JZ   LOOP         ;IF = 0, INVALID KEY
 75 002B C420               LDI   X'20
 76 002D 01     LOOP1:      XAE                ;ROW DRIVER TO E REG.
 77 002E C380                LD   -128(3)      ;READ ROW INTO ACCUM
 78 0030 9C0C               JNZ   SHIFT        ;IF NOT 0, VALID KEY
 79 0032 C203                LD   SWITCH(2)
 80 0034 F408               ADI   8            ;INCR SWITCH BY 8
 81 0036 CA03                ST   SWITCH(2)
 82 0038 01                 XAE                ;ROW DRIVER INTO ACCUM
 83 0039 1C                  SR
 84 003A 98E0                JZ   OVER         ;NO KEY FOUND
 85 003C 90EF               JMP   LOOP1
 86 003E 1C     SHIFT:       SR
 87 003F 9806                JZ   RELEAS       ;IF = 0, KEY DECODED
 88 0041 01                 XAE                ;SAVE CODE IN E REG
 89 0042 AA03               ILD   SWITCH(2)    ;INCR SWITCH VALUE
 90 0044 01                 XAE                ;RECALL FOR NEXT SHFT
 91 0045 90F7               JMP   SHIFT
 92 0047 C33F   RELEAS:      LD   ALLKYS(3)    ;READ KEY MATRIX
 93 0049 01                 XAE                ;SAVE CODE
 94 004A 8F05               DLY   5            ;DEBOUNCE 5 MS
 95 004C C33F                LD   ALLKYS(3)
 96 004E 50                 ANE                ;COMPARE NEW WITH OLD
 97 004F 9CF6               JNZ   RELEAS       ;IF 0, KEY RELEASED
 98 0051 C201                LD   TEMPL(2)
 99 0053 33                 XPAL  3
100 0054 C202                LD   TEMPH(2)
101 0056 37                XPAH  3            ;RESTORE PTR 3
102 0057 C203                LD   SWITCH(2)    ;GET KEYCODE
103 0059 01                 XAE                ;SAVE CODE IN E REG
104 005A 3F                 XPPC  3            ;RETURN TO CALLING
105                                            ;PROGRAM
106 005B 90B3                JMP SCAN
107
108
```

Figure 2C2–3 (Continued)

```
109
110
111                ;DATA AREA
112
113
114         0300   RAM = X'0300
115
116         0000   SAVE = 0
117
118         0001   TEMPL = 1
119
120         0002   TEMPH = 2
121
122         0003   SWITCH = 3
123
124         0004   KEYMD = 4
125
126         003F   ALLKYS = X'3F
127
128         0900   PERIPH = X'0900
129
130
131         0000                      END


ALLKYS   003F          KEYMD    0004 *        LOOP     0021
LOOP1    002D          OVER     001C          PERIPH   0900
RAM      0300          RELEAS   0047          SAVE     0000
SCAN     0010          SHIFT    003E          START    000D *
SWITCH   0003          TEMPH    0002          TEMPL    0001



NO ERROR LINES
SOURCE CHECKSUM=557B

***DISC SECTORS USED***

FIRST INPUT SECTOR HEX -  0292
FINAL INPUT SECTOR HEX -  0296
```

NS10546

Figure 2C2–3 (Concluded)

USING SC/MP WITH A KEYBOARD (20-KEY) ENCODER

General Description

The keyboard matrix and the 20-key encoder shown in figure 2C2-4 can be used with SC/MP to provide continuous keyboard scanning or the keyboard can be used as an interrupt device. For continuous scanning, the 'Sense B' input to SC/MP is tested for a 'logic 1', and if this condition is detected, the 5-bit keycode is stored in RAM and the program halts. In a real-life system, the code could be saved in the Extension Register and control could be transferred to a user routine that processes the keycode data.

When the keyboard is used as an interrupt device, SC/MP executes a 'main' program until a 'logic 1' (keycode input available) is detected at the 'Sense A' input. The program then jumps to an interrupt service routine, which inputs the code, saves it in the Extension Register, and returns control to the interrupted program.

System Operation

In figure 2C2-1 where SC/MP is used as a keyboard scanner, the binary code for each key is computed by the program. As shown in figure 2C2-4, the TRI-STATE CMOS encoder (MM74C923) provides all of the key-encoding logic, and, in addition, it provides switch debouncing and a 2-key rollover function. The 2-key rollover guarantees that the Data Available Signal at pin 13 goes from a logic '1' (upon valid key entry) to a logic '0' – even though a second key is depressed before the first 1-to-0 transition is completed. A logic '1' for the second key will then appear at pin 13 after some predetermined debounce interval. Operation of the key encoder system is very simple and straightforward. When closure of a key contact is detected, the encoder debounces the key and loads the appropriate binary code into five TRI-STATE output latches; the presence of this output data is indicated by driving the Data Available Signal high. As shown, the Data Available Pin is connected to the Sense B input of SC/MP or, if the keyboard is interrupt-driven, to Sense A. In either case, SC/MP responds by outputting the assigned keyboard address, and at read-strobe (NRDS) time, pin 14 (the Output Enable Signal) of the encoder is driven low. Subsequently, the latched keycode data are read into the Accumulator.

Software Considerations

Minimum software support for the keyboard encoder shown in figure 2C2-4 includes a scan (or interrupt service) routine for inputting data to SC/MP. In more-sophisticated software systems, the program, in addition to the basic scan function, may include recognition code for any combination of hex/command keys and also some sort of debug code to ensure valid processing of the input data. Some software examples are given in figures 2C2-5 and 2C2-6; a summary of these programs follows.

Figure 2C2-5

This program uses the Sense B input for keyboard interrogation and uses the Extension Register to index the keys; the keycode data are stored in memory with no processing involved.

Figure 2C2-6

This program uses Sense A as a keyboard interrupt; the interrupt service routine gets the keycode, saves it in the Extension Register, and then returns to the main program.

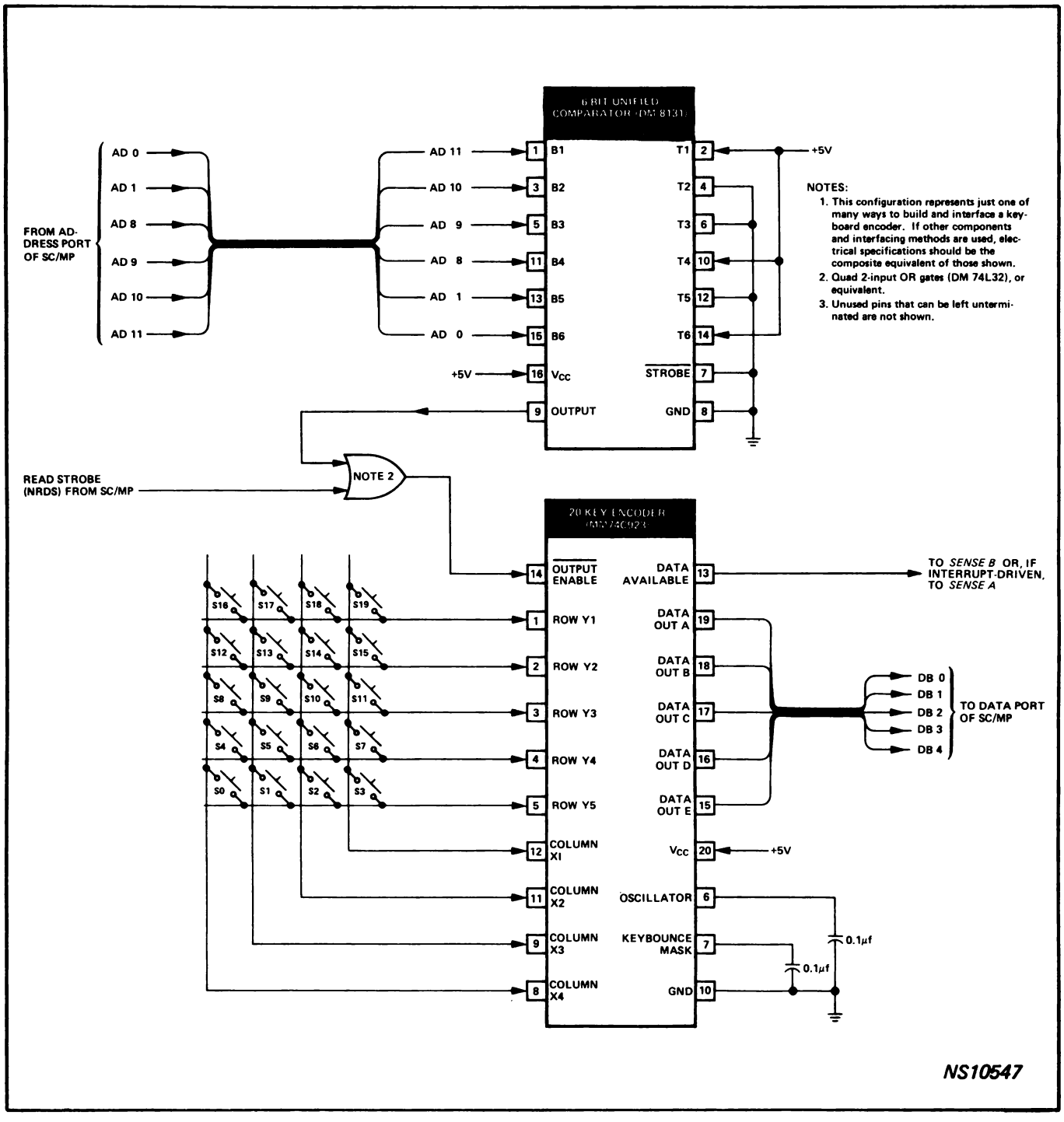
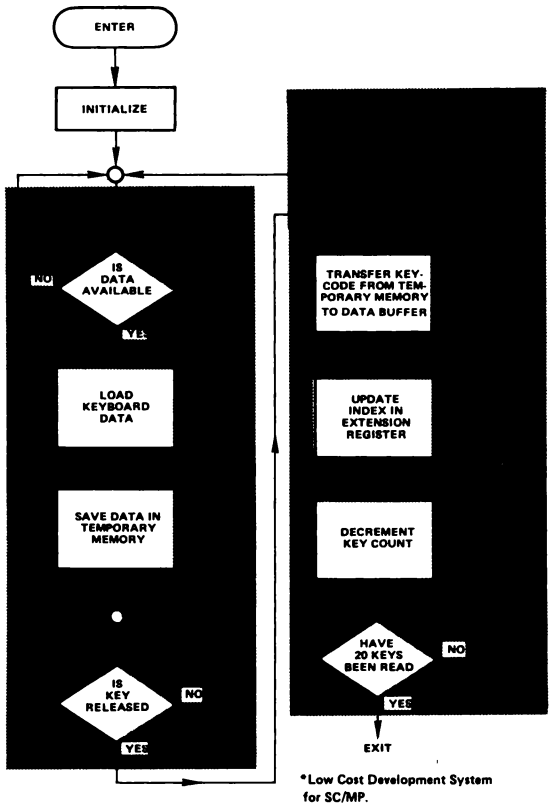


Figure 2C2–4. Using SC/MP with A Keyboard (20-Key) Encoder

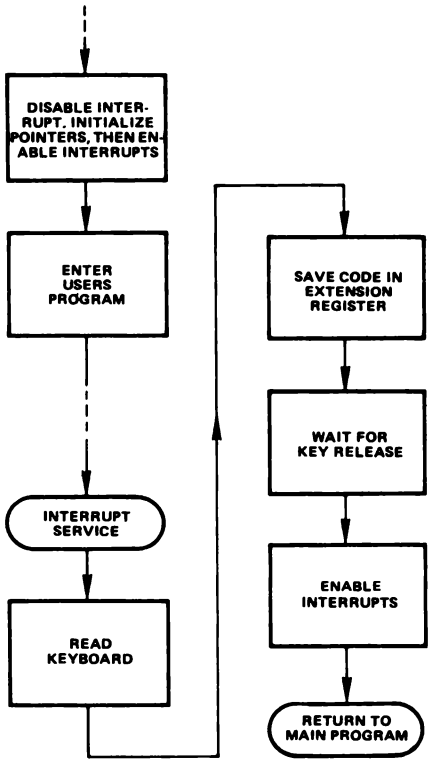



```
 1                              TITLE SCAN, ' 20 KEY KYBD SCAN '
 2
 3
 4
 5                      ;THIS PROGRAM IS USED WITH THE 74C923 20 KEY
 6                      ;KEYBOARD ENCODER.  THE PORTION OF THE PROGRAM
 7                      ;LABELED MAIN IS USED AS A MEANS TO GET TO
 8                      ;THE KEYBOARD SCAN ROUTINE.
 9                      ;SENSE B IS CONNECTED TO THE DATA VALID
10                      ;SIGNAL OF THE 74C923.
11                      ;20 KEY CODES ARE STORED IN MEMORY.
12                      ;THE ROUTINE THEN EXITS TO USER PROGRAM.
13
14
15
16
17 0000 08              NOP
18 0001 04              DINT
19 0002 C400            LDI   L(RAM)
20 0004 32              XPAL  2
21 0005 C403            LDI   H(RAM)
22 0007 36              XPAH  2             ;PTR 2 = DATA BUFFER
23 0008 C414            LDI   X'14
24 000A CA00            ST    (2)           ;STORE COUNT
25 000C 05              IEN                 ;ENABLE INTERRUPT
26 000D C402            LDI   2
```

Figure 2C2–5. Using Sense B of SC/MP to Input Keycode Data – Flowchart and Program Listing

```
27 000F 01             XAE              ;SET E REGISTER = 2
28 0010 C401  ENTRY:   LDI  L(KYBD)
29 0012 31             XPAL 1
30 0013 C409           LDI  H(KYBD)
31 0015 35             XPAH 1           ;POINTER 1 = KEYBOARD ADDRESS
32
33
34                     ;THE NEXT SECTION OF CODE STARTING WITH
35                     ;THE LABEL 'SCAN' AND ENDING WITH THE
36                     ;INSTRUCTION JNZ RELEASE IS THE ACTUAL
37                     ;TESTING FOR KEYBOARD INPUT AND KEY RELEASE.
38
39
40
41
42 0016 06     SCAN:   CSA
43 0017 D420           ANI  X'20        ;MASK TO TEST  SENSE B INPUT
44 0019 9C02           JNZ  INPUT       ;SENSE B = 1, VALID KEY
45 001B 90F9           JMP  SCAN        ;SENSE B = 0, CONTINUE
46 001D C100  INPUT:   LD   (1)         ;GET KEYBOARD
47 001F D41F           ANI  X'1F        ;BLANK HI 3
48 0021 CA01           ST   1(2)        ;SAVE KEYCODE
49 0023 06     RELEAS: CSA
50 0024 D420           ANI  X'20        ;MASK TO TEST  SENSE B
51 0026 9CFB           JNZ  RELEAS      ;IF ACCUM = 1, LOOP UNTIL
52                                      ;KEY RELEASED
53 0028 C201  MAIN:    LD   1(2)        ;GET KEYCODE
54 002A CA80           ST   -128(2)     ;SAVE KEYCODE IN DATA BUFF.
55 002C 02             CCL
56 002D 40             LDE
57 002E F401           ADI  1
58 0030 01             XAE              ;E REGISTER UPDATED FOR USE
59                                      ;AS INDEX ON DATA BUFFER
60 0031 BA00           DLD  (2)         ;DECREMENT COUNT
61 0033 9CE1           JNZ  SCAN
62             EXIT:                     ;AFTER 20 KEYS TAKEN CONTINUE
63                                       ;USER PROGRAM
64
65
66       0901  KYBD = X'0901
67       0300  RAM  = X'0300
68
69
70       0000                END

ENTRY   0010 *       EXIT    0035 *       INPUT   001D
KYBD    0901         MAIN    0028 *       RAM     0300
RELEAS  0023         SCAN    0016

NO ERROR LINES
SOURCE CHECKSUM=81EB                                    NS10548
```

Figure 2C2–5 (Concluded)




```
 1                    TITLE SCAN1, 'SC/MP TO 20 KEY KEYBOARD'
 2
 3
 4          ;THIS PROGRAM USES SENSE A AS AN INTERRUPT TO
 5          ;SC/MP, AND WORKS WITH THE 74C923 KEYBOARD ENCODER.
 6          ;THE KEY CODE IS NOT PROCESSED
 7          ;DEPRESSION OF ANY KEY CAUSES AN INTERRUPT
 8          ;THE PROCESSOR THEN GETS THE KEY CODE.
 9
10
11          ;INITIALIZE AND WAIT FOR INTERRUPT
12
13 0000 08      NOP
14 0001 04      DINT                    ;DISABLE INTERRUPT
15 0002 C410    LDI     L(KEYSAV)-1
16 0004 33      XPAL    3
17 0005 C400    LDI     H(KEYSAV)
18 0007 37      XPAH    3               ;PTR 3 = INTERRUPT SERVICE
19 0008 C401    LDI     L(KYBD)
20 000A 31      XPAL    1
21 000B C409    LDI     H(KYBD)
22 000D 35      XPAH    1               ;PTR 1 = KEYBOARD ADDRESS
23 000E 05      IEN                     ;ENABLE INTERRUPT
24
25
26
```

Figure 2C2–6. Using Keyboard as Interrupt Device (via Sense A) – Flowchart and Program Listing

```
27                          ;THE NEXT  INSTRUCTION  SIMULATES A
28                          ;USERS MAIN PROGRAM.   IN THIS CASE THE
29                          ;PROGRAM IS WASTING TIME WAITNG FOR
30                          ;AN INTERRUPT FROM THE KEYBOARD.
31
32
33
34 000F 90FE   LOOP:    JMP      LOOP          ;AWAITING INTERRUPT
35
36
37                      ;THE FOLLOWING CODE IS THE INTERRUPT
38                      ;SERVICE.   THIS ROUTINE GETS THE KEYCODE,
39                      ;SAVES IT IN THE E REGISTER AND RETURNS
40                      ;TO THE MAIN PROGRAM.
41
42
43
44 0011 C100   KEYSAV:  LD       (1)       ;GET KEYCODE
45 0013 D41F            ANI      X'1F      ;BLANK HI 3
46 0015 01              XAE                ;SAVE CODE IN E REG
47 0016 06     RELEAS:  CSA
48 0017 D410            ANI      X'10      ;MASK TO TEST  SENSE A
49 0019 9CFB            JNZ      RELEAS    ;WAIT FOR KEY RELEASE
50
51                      ;AT THIS POINT THE USER SHOULD SAVE
52                      ;THE KEYCODE IN TEMP STORAGE, OR
53                      ;CONTINUE TO PROCESS THE CODE IN
54                      ;THE INTERRUPT ROUTINE
55
56
57
58 001B C702        LD       @2(3)         ;MODIFY P3 TO SKIP LOOP
59                                         ;AND CONTINUE MAIN PROG.
60 001D 05         IEN                    ;ENABLE INTERRUPT
61 001E 3F         XPPC     3             ;RETURN TO MAIN PROG.
62                                         ;WITH KEYCODE IN E REG.
63 001F 90F0       JMP      KEYSAV
64
65
66       0901  KYBD     = X'0901
67
68       0000            END
SCAN1 SC/MP TO 20 KEY KEYBOARD

KEYSAV  0011          KYBD    0901          LOOP    000F
RELEAS  0016




NO ERROR LINES
SOURCE CHECKSUM=8D75
```

NS10549

Figure 2C2–6 (Concluded)

USING SC/MP WITH THE MM5740 (90-KEY) ENCODER

General Description

Figure 2C2-7 shows one way in which SC/MP might be interfaced to a relatively large key matrix. This particular scheme uses a 90-key encoder with supporting circuit peripherals, a unified bus comparator for address assignment, and a TRI-STATE octal output buffer. The encoder is capable of providing a 9-bit output code; however, in figure 2C2-7, only 8 bits are used – DB0 through DB6 for keycode information and DB7 for parity. The encoder also provides internal switch debouncing and a 2-key (or N-key) rollover function.

System Operation

Clock requirements for the encoder are supplied by an LM555 oneshot; this circuit provides a suitable operating frequency. The shift, shift lock, and special-character control functions are implemented, respectively, by switches S1, S2, and S3; a shift-lock indicator is provided also. The encoder is operated in the pulse data strobe mode; hence, when valid data are entered by the keyboard, the valid-data 'flag' (pins 13/14) is latched for timing compatibility with the Sense A (interrupt mode) or the Sense B (scan mode) inputs. For either operating mode, SC/MP responds by outputting the assigned keyboard address, and at read-strobe (NRDS) time, the output of the TRI-STATE buffer is enabled. Accordingly, the keycode data (BD0 through BD7) is read into SC/MP.

Software Considerations

Software requirements for the MM5740 system could be similar to those shown in figure 2C2-5 (scan mode) and figure 2C2-6 (interrupt mode). In some applications, the keycode data can be interpreted by more than one program. For instance, one program might implement a quality control function, another could use the same data for display purposes, while a third program could use the data for statistical analysis.

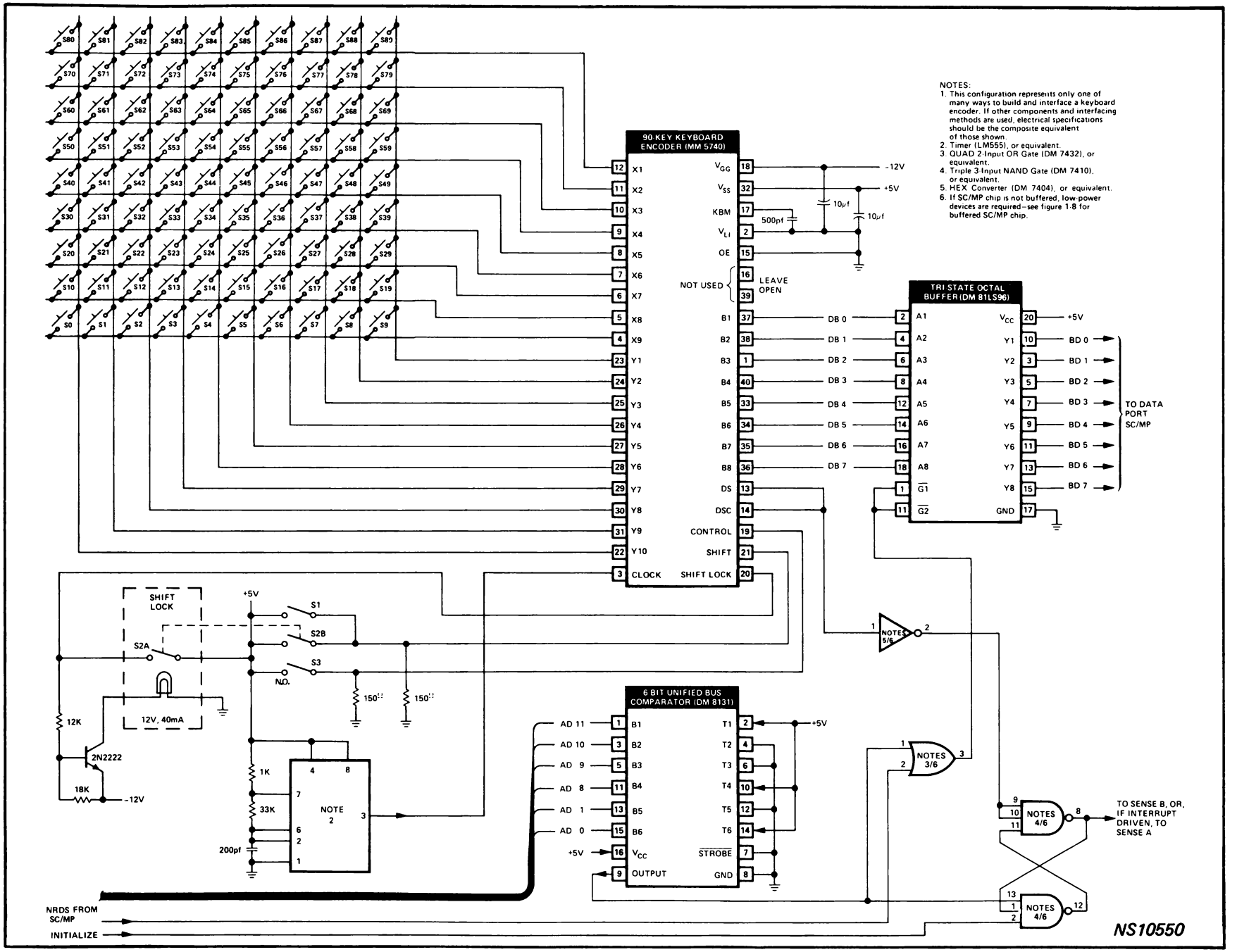


Figure 2C2–7. Interfacting SC/MP with the MM5740 (90-Key) Encoder

AN INTERRUPT-DRIVEN KEYBOARD/DISPLAY SYSTEM

General Description

The preceding applications (figures 2C2-1, 2C2-4, and 2C2-7) show how SC/MP can be used to input and process data from a keyboard. This application shows how SC/MP can be used to develop a functional keyboard/display system that is interrupt-driven; a block diagram of the system is shown in figure 2C2-8. Besides SC/MP, the system hardware consists of a 20-key matrix (16 hexadecimal and 4 command keys), a 20-key encoder, a 6-digit/7-segment LED display that is software multiplexed, and appropriate memory, decode, and buffer/driver devices. System software consists of a monitor program that allows user to *read* or *modify* (write into) a memory location, to *execute* a program starting at any address, and to *abort* the interrupt service at any time. Four main subroutines are callable from the keyboard: SCAN, MUXDIS, G4HEX, and G2HEX. The SCAN subroutine gets valid keycode data from the 20-key matrix; also, this subroutine automatically calls the MUXDIS subroutine, which services the LED display. The G4HEX and G2HEX subroutines also call up SCAN and get, respectively, four hexadecimal characters and two hexadecimal characters from the keyboard. The hexadecimal characters are stored in a temporary table (six consecutive memory locations) and then are packed in three contiguous memory locations with the following assignments: HIGH ADDRESS, LOW ADDRESS, and DATA.

Memory assignments for the entire keyboard/display system are shown in the memory map of figure 2C2-8. The foregoing monitor program is resident in the 512-by-8 PROM; an additional 30 words of RAM also are required for this program. Memory services for the keyboard and display peripherals are as shown.

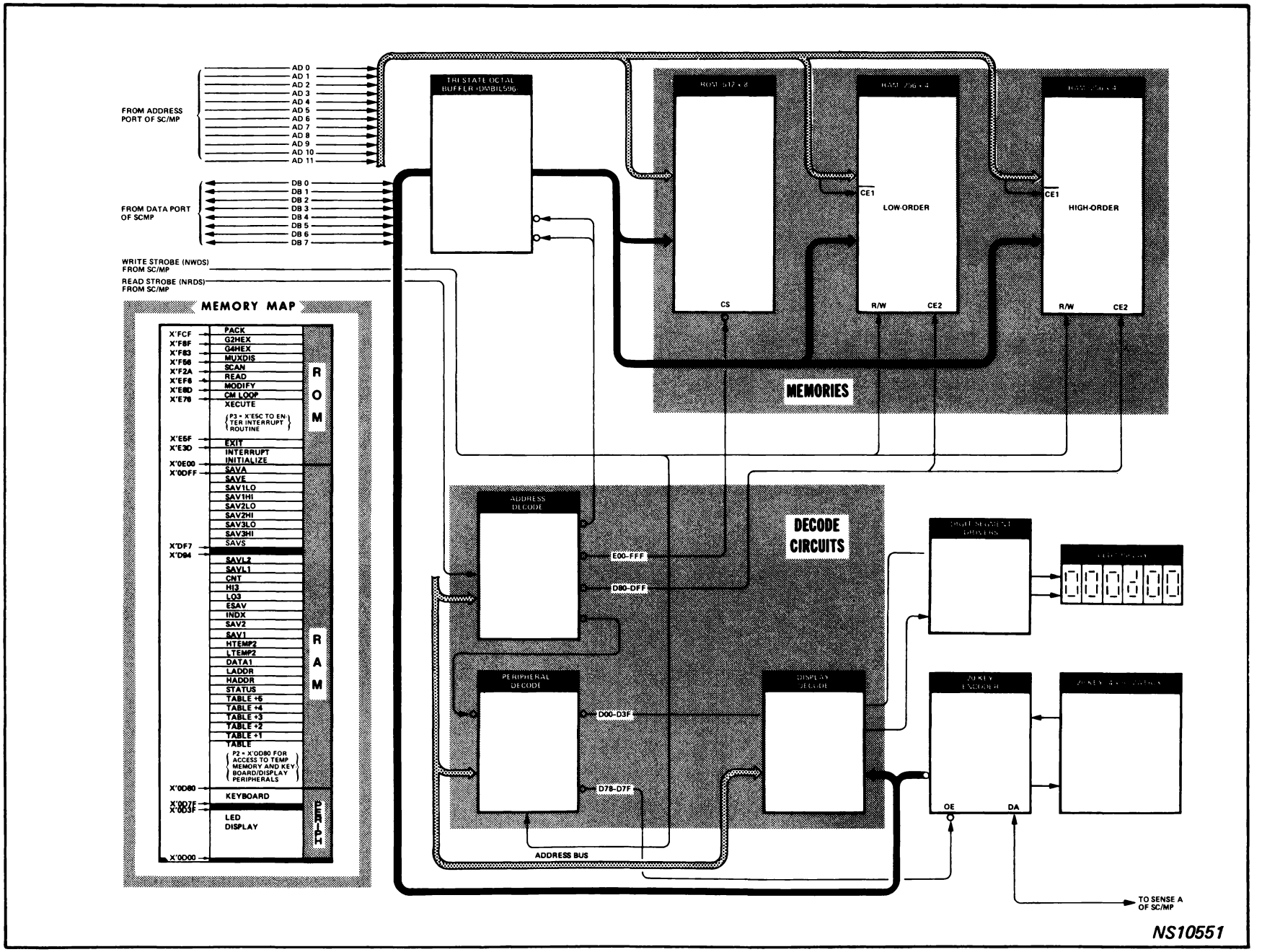


Figure 2C2–8. Interrupt-Driven Keyboard/Display System–Block Diagram and Memory Map

Any one of the 20 keys can be pressed to initiate an interrupt, and when this condition occurs, the six LED indicators display a 00d000 as an 'interrupt-recognition' code.

NOTE

Although the depression of any key causes an interrupt, the interrupt will automatically *abort* if the key is not a legitimate command – thus, the operator may not see the initial message 00d000.

Once the interrupt is accepted, the keyboard/display system can be utilized in any one of three modes – READ, MODIFY, and EXECUTE; a fourth mode (ABORT) is used to return control to the interrupted program. The first key entered must be a legitimate READ, MODIFY, or EXECUTE command; if the key entry is invalid, or if a valid command is followed by an illegal sequence of keys, the interrupt service is exited just as though the 'ABORT' key were pressed.

Operating procedures and a summary of results for each command key are given below.

READ Mode

1. Depress and release READ key.

2. Enter address by 'press-and-release' of any four hexadecimal keys; display will show 4-digit hexadecimal address and 2-digit hexadecimal data in that address. After address is entered, it can be incremented and display updated by successively pressing READ key.

3. To modify a memory location while in read mode, proceed as follows:

   Press MODIFY key and then whichever two hexadecimal keys are required for data modification. Before new data are entered, display will show hexadecimal address and 'old' data; after modification, display will show same address and 'new' data.

4. Return to interrupted program by 'press-and-release' of ABORT key.

MODIFY Mode

1. Depress and release MODIFY key.

2. Enter address by 'press-and-release' of any four hexadecimal keys; enter data in selected address by 'press-and-release' of any two hex keys. Before new data is entered, display will show hexadecimal address and 'old' data; after modification, display will show same address and 'new' data.

3. After address is entered, it can be incremented and display updated by successively pressing MODIFY key.

NOTE

If, after entry of each modify command, user fails to enter 2-key hexadecimal data, interrupt service is terminated and control is returned to interrupted program.

4. To transfer from modify mode to read mode, simple 'press-and-release' READ key after completion of any legitimate modify command.

XECUTE Mode

1. Depress and release XECUTE key.

2. Press and release four hexadecimal keys to specify point-of-entry address and to transfer control to user's program.

NOTE

Entry errors for first three keys can be nullified by pressing ABORT and then repeating operating sequence for XECUTE. Once fourth key is entered and address is in error, system may require reinitialization.

ABORT Mode

1. Depress and release ABORT key.

2. Interrupt service is exited and control is transferred to interrupted program.

Software Considerations

The following entry and subroutine-calling codes (or their functional equivalents) must be implemented in the users program if the keyboard/display is used for anything other than a passive monitor.

1. Entry Code to INTERRUPT SERVICE Routine
2. Calling Code for MUXDIS Subroutine
3. Calling Code for Keyboard SCAN Routine
4. Calling Code for G4HEX Routine
5. Calling Code for G2HEX Routine

Coding schemes to satisfy each of the preceding requirements are described in the listings that follow. Figures 2C2-9 through 2C2-12 that follow the listings provide a software flowchart for each subroutine and other relevant code; figure 2C2-13 provides a complete printout of the monitor program.

System Schematics

Pin-to-pin wiring of the keyboard/display system is shown schematically in figure 2C2-14.

Purpose: Calling Interrupt Service Routine via Pointer 3.

Conditions: Following Code must appear somewhere in users program.

Code:

```
    •
    •                          ; { USERS PROGRAM
    •
LDI     X'FF        ;GET LOW-ORDER BYTE OF INTERRUPT ADDRESS
XPAL    3           ;PUT IN POINTER REGISTER 3
LDI     X'D         ;GET HIGH-ORDER BYTE OF INTERRUPT ADDRESS
XPAH    3           ;PUT IN POINTER REGISTER 3
                    ;POINTER 3 NOW LOADED WITH '1-LESS' THAN THE ADDRESS OF
                    ;THE INTERRUPT SERVICE ROUTINE
IEN                 ;ENABLE INTERRUPT
    •
    •               ;CONTINUE USERS PROGRAM
    •               ;WHEN ANY KEY IS DEPRESSED, USERS PROGRAM IS
                    ;INTERRUPTED AND KEYBOARD MONITOR PROGRAM IS
                    ;EXECUTED
```

Purpose: Call MUXDIS Subroutine.

Conditions: Locations X'D80 through X'D85 must be preloaded with the information to be displayed the hex digit in X'D80 and least significant digit in X'D85; Pointer 2 must be equal to X'D80. This subroutine saves the contents of calling Pointer 1 upon entry and restores this pointer upon exit.

Code:

```
    •
    •                          ; { USERS PROGRAM
    •
DINT                ;DISABLE INTERRUPT
LDI     X'55        ;GET LOW-ORDER BYTE OF MUXDIS SUBROUTINE ADDRESS
XPAL    1           ;PUT LOW-ORDER BYTE IN POINTER 1
LDI     X'F         ;GET HIGH-ORDER BYTE OF MUXDIS SUBROUTINE ADDRESS
XPAH    1           ;PUT HIGH-ORDER BYTE IN POINTER 1
                    ;POINTER 1 NOW LOADED WITH '1-LESS' THAN THE ADDRESS
                    ;OF THE MUXDIX SUBROUTINE
XPPC    1           ;CALL MUXDIS SUBROUTINE
RETURN: <--------->  ;INSTRUCTION TO BE EXECUTED UPON RETURN FROM MUXDIS
IEN                 ;ENABLE INTERRUPT SO MONITOR PROGRAM CAN BE EXECUTED
                    ;UPON ENTRY FROM KEYBOARD
    •
    •               ; { CONTINUE USERS PROGRAM
    •
```

Purpose: Calling Keyboard SCAN Subroutine via Pointer 3.

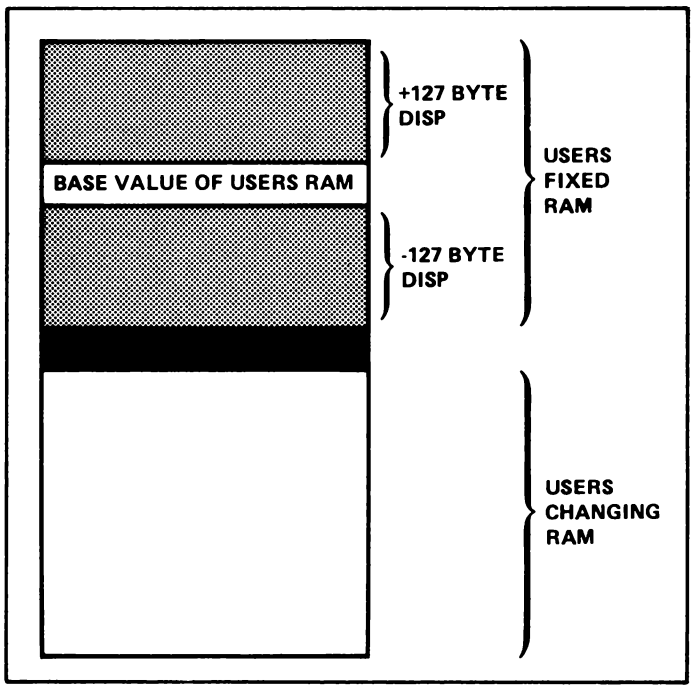
Conditions: Pointer 2 must be loaded with X'D80. If Pointer 2 is already being used as the RAM pointer, the following code is required to implement the subroutine call.

Code:

```
DINT                    ;DISABLE INTERRUPT
LDI     H(USER RAM BASE) ;GET HIGH-BASE RAM ADDRESS IN ACCUMULATOR
XPAH    2               ;PUT HIGH-BASE RAM ADDRESS IN POINTER 2 AND 'OLD'
                        ;P2-HIGH IN ACCUMULATOR
XAE                     ;TEMPORARILY SAVE 'OLD' P2-HIGH IN EXTENSION
LDI     L(USER RAM BASE) ;GET LOW-BASE RAM ADDRESS IN ACCUMULATOR
XPAL    2               ;PUT LOW-BASE RAM ADDRESS IN POINTER 2 AND 'OLD'
                        ;P2-LOW IN ACCUMULATOR
ST      P2SAV(2)        ;SAVE 'OLD' P2-LOW IN P2SAV
LDE                     ;PUT 'OLD' P2-HIGH IN ACCUMULATOR
ST      P2SAV+1(2)      ;SAVE 'OLD' P2-HIGH IN P2SAV+1 WHERE 'P2SAV' AND
                        ;'P2SAV+1' ARE RESERVED RAM LOCATIONS WITHIN
                        ;BASE-VALUE DISPLACEMENT RANGE OF USER RAM
LDI     X'29            ;LOAD ACCUMULATOR WITH '1-LESS' THAN LOW-ORDER
                        ;ADDRESS OF SCAN ROUTINE
XPAL    3               ;PUT LOW-ORDER ADDRESS OF SCAN IN POINTER 3 AND
                        ;OLD' P3-LOW IN ACCUMULATOR
ST      P2SAV+2(2)      ;SAVE 'OLD' P3-LOW IN P2SAV+2(2)
LDI     X'F             ;GET HIGH-ORDER ADDRESS OF SCAN ROUTINE
XPAH    3               ;PUT HIGH-ORDER ADDRESS OF SCAN IN POINTER 3 AND
                        ;'OLD' P3-HIGH IN ACCUMULATOR
ST      P2SAV+3(2)      ;SAVE 'OLD' P3-HIGH IN P2SAV+3 WHERE 'P2SAV+2' AND
                        ;'P2SAV+3' ARE RESERVED RAM LOCATIONS WITHIN
                        ;BASE-VALUE DISPLACEMENT RANGE OF USERS RAM
LDI     X'80            ;GET LOW-ORDER RAM-BASE ADDRESS FOR SCAN
XPAL    2               ;PUT LOW-ORDER ADDRESS IN POINTER 2
LDI     X'D             ;GET HIGH-ORDER RAM-BASE ADDRESS FOR SCAN
XPAH    2               ;PUT HIGH-ORDER ADDRESS IN POINTER 2
XPPC    3               ;CALL SCAN ROUTINE
<       >               ;RETURN FROM SCAN WITH KEYCODE IN EXTENSION REGISTER
LDI     H(USER RAM BASE) ;GET HIGH-BASE RAM ADDRESS IN ACCUMULATOR
XPAH    2               ;RESTORE HIGH-BASE RAM
LDI     L(USER RAM BASE) ;GET LOW-BASE RAM ADDRESS IN ACCUMULATOR
XPAL    2               ;RESTORE LOW-BASE RAM
LDE                     ;GET KEYCODE IN ACCUMULATOR
ST      ESAV(2)         ;SAVE KEYCODE IN USERS RAM
LD      P2SAV+3(2)      ;GET 'OLD' HIGH-ORDER CONTENT OF POINTER 3
XPAH    3               ;RESTORE 'OLD' P3-HIGH
LD      P2SAV+2(2)      ;GET 'OLD' LOW-ORDER CONTENT OF POINTER 3
XPAL    3               ;RESTORE 'OLD' P3-LOW
LD      P2SAV+1(2)      ;GET 'OLD' LOW-ORDER CONTENT OF P2
XAE                     ;TEMPORARILY SAVE 'OLD' P2-LOW IN EXTENSION
LD      P2SAV(2)        ;GET 'OLD' HIGH-ORDER CONTENT OF P2
XPAH    2               ;RESTORE 'OLD' HIGH-ORDER CONTENT OF POINTER 2
LDE                     ;GET 'OLD' P2-LOW FROM EXTENSION
XPAL    2               ;RESTORE 'OLD' LOW-ORDER CONTENT OF POINTER 2
                        ;CONTENTS OF P2 IS NOW THE SAME AS IT WAS BEFORE
                        ;CALLING SCAN ROUTINE
 •                      ;CONTINUE USERS PROGRAM
 •                      ;
 •                      ;
```

Purpose: How to save and restore pointers when calling G4HEX Subroutine via Pointer 1.

Conditions: Pointer 3 must contain SCAN address and Pointer 2 must point to X'D80 (RAM base for code that follows). Locations SAV3LO/ SAV3HI must be preloaded by the user for program return in case of an entry error from keyboard. Pointer 2 is the RAM pointer for the users program; thus, a few RAM locations within displacement range of the base-value of the users fixed RAM must be reserved– see illustration.

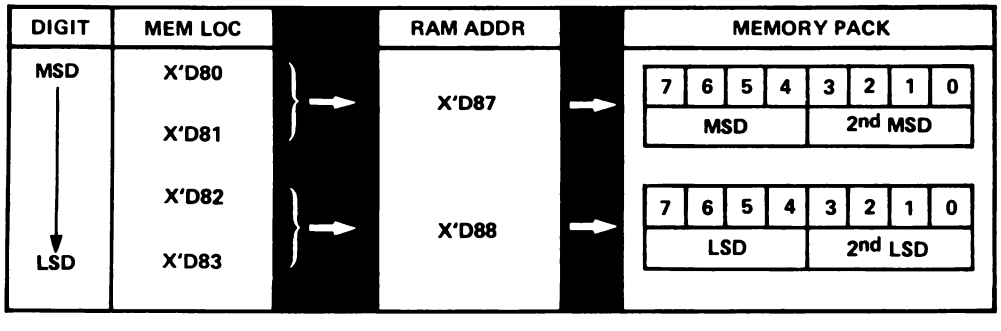


Code:

```
•
•                                   ; { USERS PROGRAM
•
DINT                                ;DISABLE INTERRUPT
LDI     H(USERS RAM BASE)           ;GET HIGH-BASE RAM ADDRESS IN ACCUMULATOR
XPAH    2                           ;PUT HIGH-BASE RAM ADDRESS IN POINTER 2 AND 'OLD'
                                    ;P2-HIGH IN ACCUMULATOR
XAE                                 ;TEMPORARILY SAVE 'OLD' P2-HIGH IN EXTENSION
LDI     L(USERS RAM BASE)           ;GET LOW-BASE RAM ADDRESS IN ACCUMULATOR
XPAL    2                           ;PUT LOW-BASE RAM ADDRESS IN POINTER 2 AND 'OLD'
                                    ;P2-LOW IN ACCUMULATOR. POINTER 2 IS NOW EQUAL
                                    ;TO USERS RAM BASE.
ST      P2SAVL(2)                   ;SAVE 'OLD' VALUE OF P2-LOW
LDE                                 ;PUT 'OLD' VALUE OF P2-HIGH IN ACCUMULATOR
ST      P2SAVH(2)                   ;SAVE 'OLD' VALUE OF P2-HIGH
LDI     X'29                        ;LOAD ACCUMULATOR WITH '1-LESS' THAN LOW-ORDER
                                    ;ADDRESS OF SCAN ROUTINE
XPAL    3                           ;PUT LOW-ORDER ADDRESS OF SCAN IN POINTER 3 AND
                                    ;'OLD' P3-LOW IN ACCUMULATOR
ST      P3SAVL(2)                   ;SAVE 'OLD' P3-LOW IN P3SAVL(2)
LDI     X'F                         ;GET HIGH-ORDER ADDRESS OF SCAN ROUTINE
XPAH    3                           ;PUT HIGH-ORDER ADDRESS OF SCAN IN POINTER 3 AND
                                    ;OLD' P3-HIGH IN ACCUMULATOR
ST      P3SAVH(2)                   ;SAVE 'OLD' P3-HIGH IN P3SAVL(2) WHERE 'P3SAVL(2)'
                                    ;AND 'P3SAVH(2)' ARE RESERVED RAM LOCATIONS WITHIN
                                    ;BASE-VALUE DISPLACEMENT RANGE OF USERS RAM
LDI     X'82                        ;LOAD ACCUMULATOR WITH '1-LESS' THAN LOW-ORDER
                                    ;ADDRESS OF G4HEX SUBROUTINE
XPAL    1                           ;PUT LOW-ORDER ADDRESS OF G4HEX IN POINTER 1 AND
                                    ;'OLD' P1-LOW IN ACCUMULATOR
ST      P1SAVL(2)                   ;SAVE 'OLD' P1-LOW
LDI     X'F                         ;GET HIGH-ORDER ADDRESS OF G4HEX SUBROUTINE
```

```
        XPAH    1               ;PUT HIGH-ORDER ADDRESS OF G4HEX IN POINTER 1
                                ;AND 'OLD' P1-HIGH IN ACCUMULATOR
        ST      P1SAVH(2)       ;SAVE 'OLD' P1-HIGH
        LDI     X'80            ;GET LOW-ORDER RAM BASE FOR G4HEX CODE
        XPAL    2               ;PUT LOW-ORDER BYTE IN POINTER 2
        LDI     X'D             ;GET HIGH-ORDER RAM BASE FOR G4HEX CODE
        XPAH    2               ;PUT HIGH-ORDER BYTE IN POINTER 2. POINTER 2
                                ;NOW CONTAINS BASE RAM ADDRESS.
        LDI     L(CALL)         ;GET LOW-ORDER CALLING ADDRESS FOR G4HEX
        ST      X'F8(2)         ;LOCATIONS X'DF8/X'DF9 ARE ESCAPE ROUTES IN CASE
                                ;OF ERROR AND MUST BE LOADED WITH THE ADDRESS OF
                                ;USERS PROGRAM WHERE THE CALL TO G4HEX OCCURS.
CALL:   XPPC    1               ;CALL G4HEX SUBROUTINE
        DINT                    ;DISABLE INTERRUPT
        LDI     H(USER RAM BASE) ;RETURN FROM G4HEX
        XPAH    2
        LDI     L(USER RAM BASE)
        XPAL    2               ;POINTER 2 NOW EQUAL TO BASE ADDRESS OF USERS RAM
        LD      P3SAVL(2)       ;GET 'OLD' LOW-ORDER CONTENT OF POINTER 3
        XPAL    3               ;RESTORE 'OLD' P3-LOW
        LD      P3SAVH          ;GET 'OLD' HIGH-ORDER CONTENT OF POINTER 3
        XPAH    3               ;RESTORE 'OLD' P3-HIGH
        LD      P1SAVL(2)       ;GET 'OLD' LOW-ORDER CONTENT OF P1
        XPAL    1               ;RESTORE 'OLD' P1-LOW
        LD      P1SAVH(2)       ;GET 'OLD' HIGH-ORDER CONTENT OF P1
        XPAH    1               ;RESTORE 'OLD' HIGH-ORDER CONTENT OF POINTER 1
        LD      P2SAVL(2)       ;GET 'OLD' LOW-ORDER CONTENT OF P2
        XAE                     ;TEMPORARILY SAVE 'OLD' P2-LOW IN EXTENSION
        LD      P2SAVH(2)       ;GET 'OLD' HIGH-ORDER CONTENT OF P2
        XPAH    2               ;RESTORE 'OLD' HIGH-ORDER CONTENT OF POINTER 2
        LDE                     ;GET 'OLD' P2-LOW FROM EXTENSION
        XPAL    2               ;RESTORE 'OLD' LOW-ORDER CONTENT OF POINTER 2
                                ;CONTENTS OF P2 IS NOW THE SAME AS IT WAS BEFORE
                                ;CALLING G4HEX ROUTINE
        IEN                     ;ENABLE INTERRUPT
         •                      ;CONTINUE USERS PROGRAM
         •                      ;
         •                      ;
```

Return Status: Upon return to users program, the G4HEX Subroutine has loaded and packed 4-hex characters as indicated below.



Purpose: How to save and restore pointers when calling G2HEX Subroutine.

Conditions: Setup and control of pointers 1, 2, and 3 and the escape locations are similar to their functional counterparts in parts in the preceding G4HEX subroutine.

Code: Same as G4HEX code.

Return Status: Upon return to users program, the G2HEX subroutine has loaded and packed 2-hex characters as indicated below.

DIGIT	MEM LOC		RAM ADDR		MEMORY PACK
MSD	X'D84	} →	X'D89	→	7 6 5 4 3 2 1 0
LSD	X'D85				MSD LSD

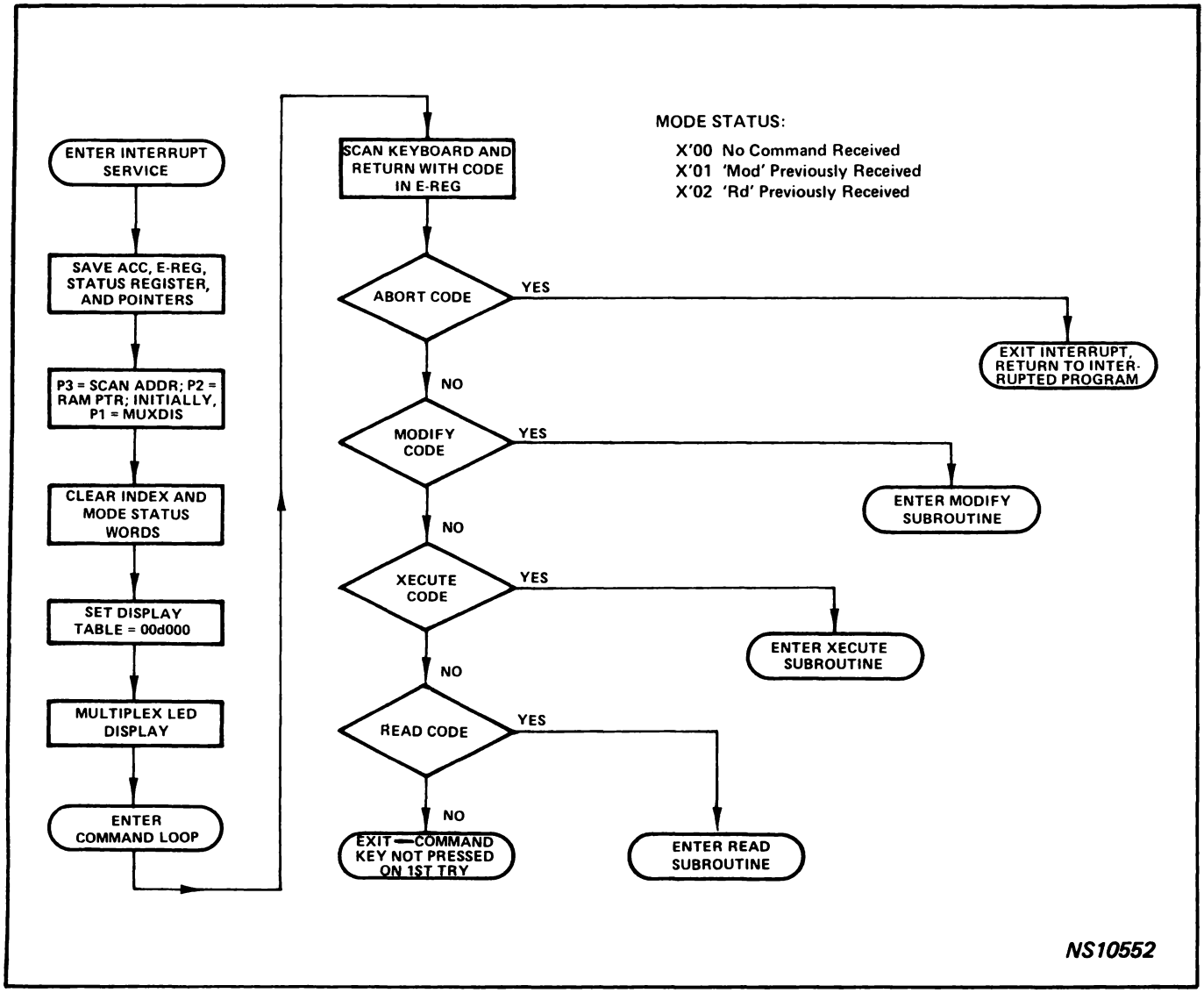


Figure 2C2–9. Flow Diagram for Interrupt Service Routine

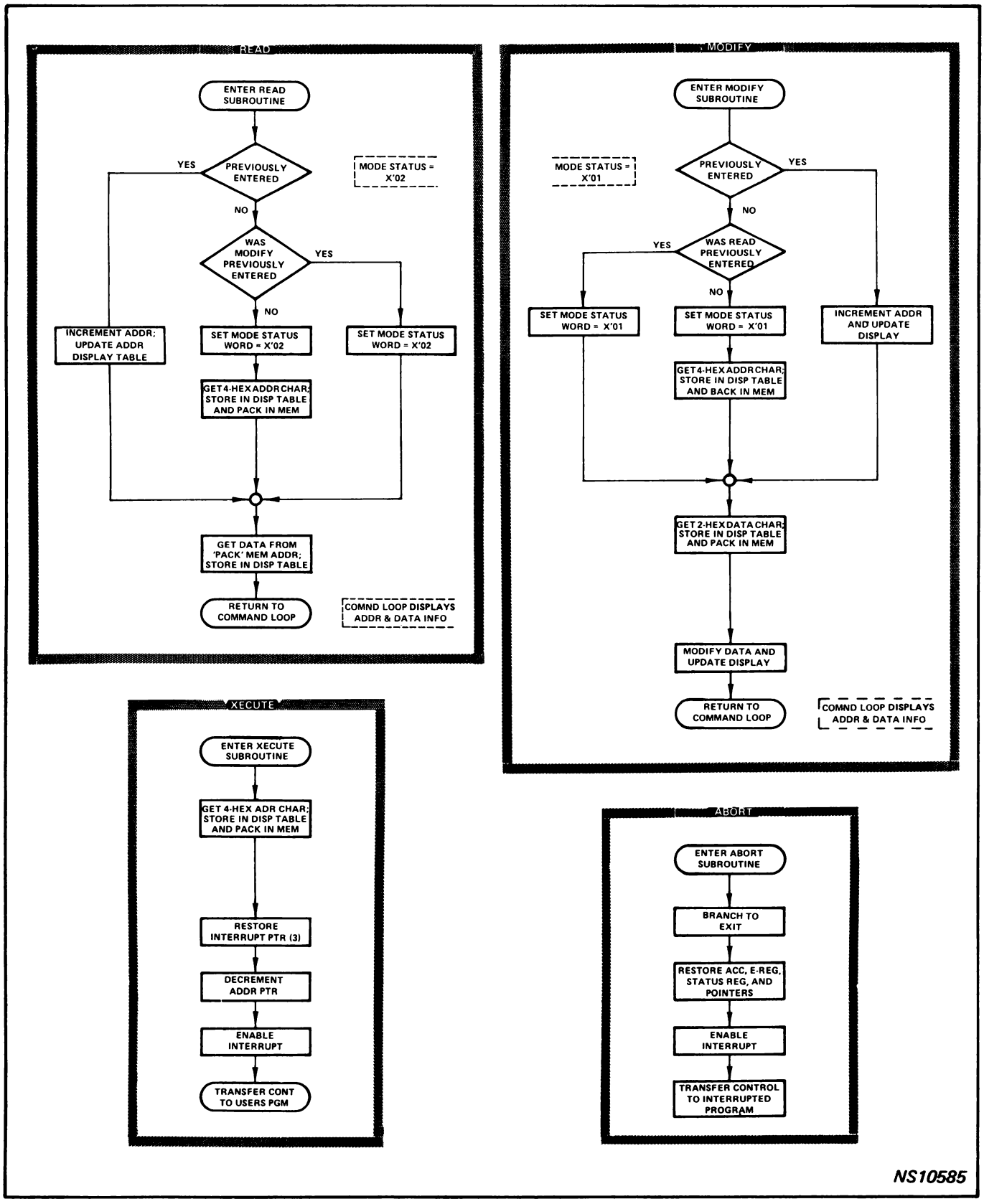


Figure 2C2–10. Flow Diagrams for READ, MODIFY, XECUTE, and ABORT Subroutines

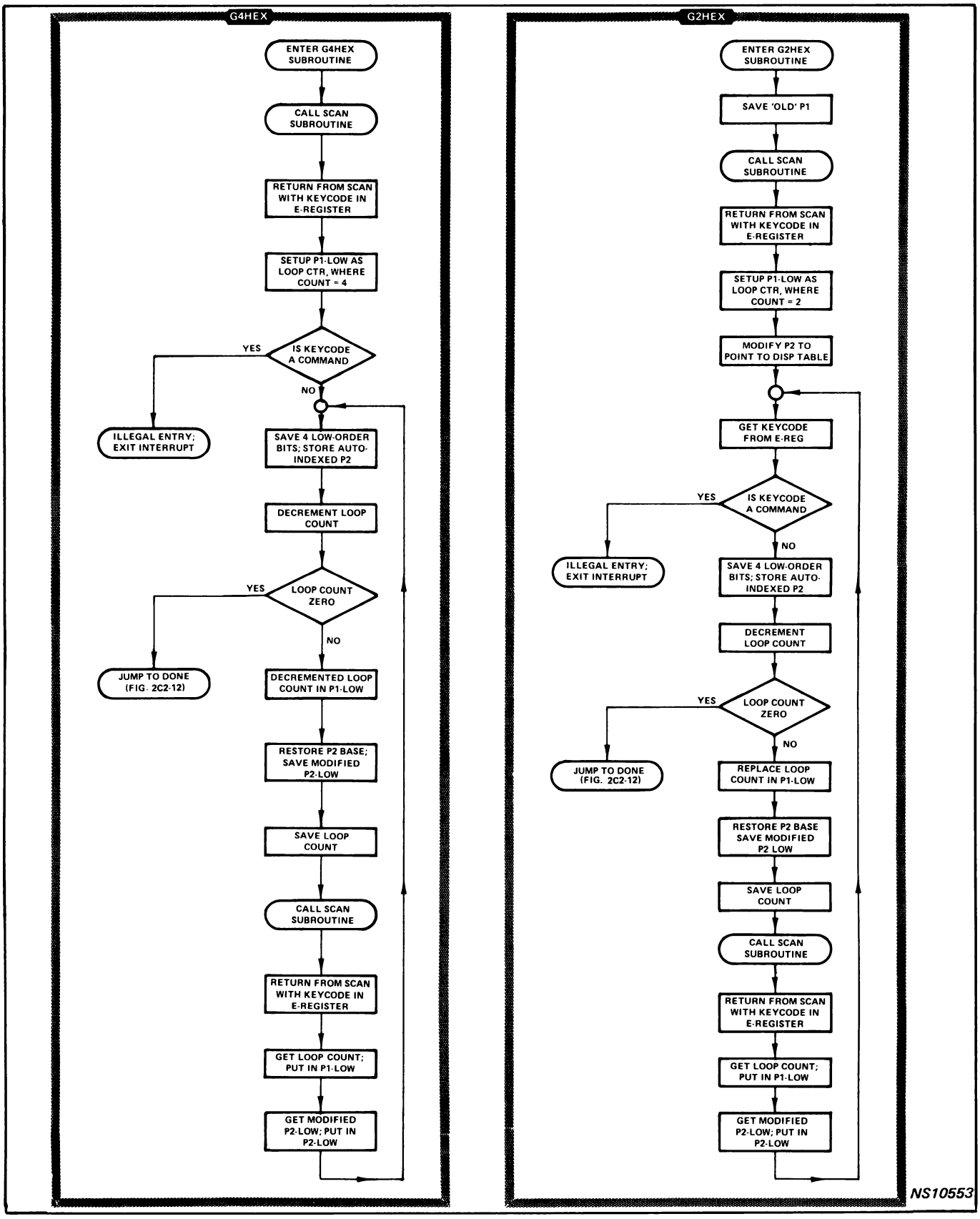


Figure 2C2–11. Flow Diagrams for G4HEX and G2HEX Subroutines

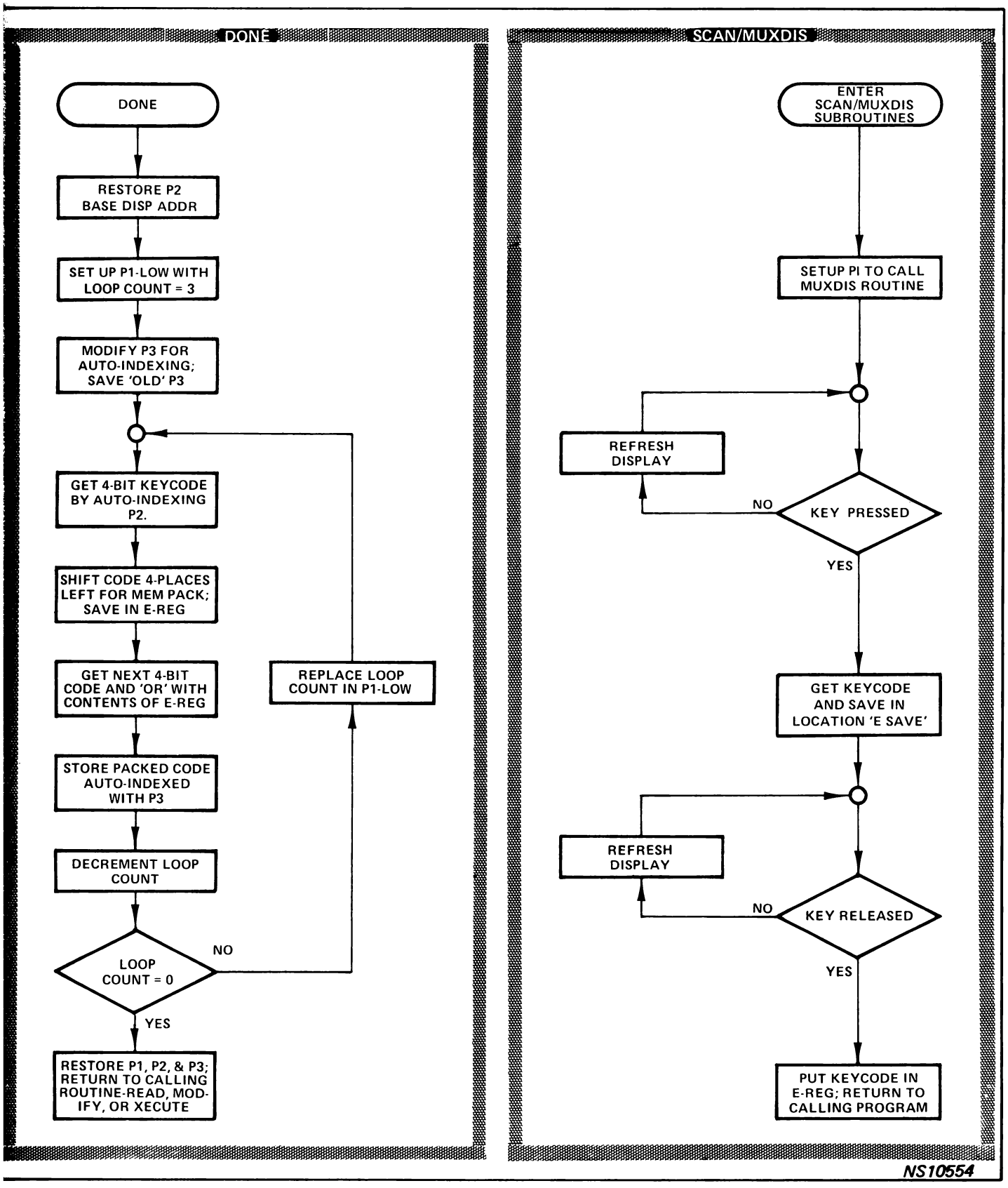


Figure 2C2–12. Flow Diagrams for SCAN/MUXDIS and DONE Subroutines

```
 1                              TITLE   KYDISI, ' KYBD & DISPLY '
 2
 3
 4
 5                      ;THIS ROUTINE IS AN INTERRUPT SERVICE.  WHEN ANY KEY
 6                      ;IS PRESSED, THE PRESENT PROGRAM IS INTERRUPTED
 7                      ;AND THE KEYBOARD & DISPLAY ARE SERVICED UNTIL AN
 8                      ;ILLEGAL ENTRY IS MADE OR THE ABORT KEY IS PRESSED.
 9                      ;A STATUS REGISTER IN RAM INDICATES WHETHER
10                      ;READ OR MODIFY MODES CHOSEN
11
12
13                      ;USERS MAIN PROGRAM MUST LOAD POINTER 3
14                      ;WITH X'0DFF OR X'0E5C AND ENSURE THAT
15                      ;INTERRUPTS ARE ENABLED WHEN KEYBOARD
16                      ;CONTROL IS DESIRED.
17
18
19
20
21
22
23
24                      ;COMMANDS
25                                  ;MODIFY = X'10
26                                  ;XECUTE = X'11
27                                  ;READ   = X'12
28                                  ;ABORT  = X'13
29
30      0E00                       =X'0E00
31
32
33                      ;LOCATIONS 'SAVE' THRU 'SAVS' ARE USED
34                      ;TO SAVE THE STATUS OF THE INTERRUPTED
35                      ;PROGRAM.
36
37
38
39      0DFF            SAVA:       = .-1
40      0DFE            SAVE:       = .-1
41      0DFD            SAV1LO:     = .-1
42      0DFC            SAV1HI: .   = .-1
43      0DFB            SAV2LO: .   = .-1
44      0DFA            SAV2HI: .   = .-1
45      0DF9            SAV3LO: .   = .-1
46      0DF8            SAV3HI: .   = .-1
47      0DF7            SAVS:   .   = .-1
48      0D80            RAM       = X'D80
49      FFFF            KYBD      = -1
50      0D00            LEDS      = X'D00
51
52
53                      ;THE NEXT GROUP OF EQUATE STATEMENTS
54                      ;DEFINE TEMPORARY WORKING RAM USED
```

Figure 2C2–13. Program Listing for Interrupt-Driven Keyboard/Display System

```
                        ;WITH POINTER 2
0000            TABLE   = 0
0006            STATUS  = 6
0007            HADDR   = 7
0008            LADDR   = 8
0009            DATA1   = 9
000A            LTEMP2  = X'A
000B            HTEMP2  = X'B
000C            SAV1    = X'C
000D            SAV2    = X'D
000E            INDX    = X'E
000F            ESAV    = X'F
0010            LO3     = X'10
0011            HI3     = X'11
0012            CNT     = X'12
0013            SAVL1   = X'13
0014            SAVL2   = X'14

0E00              = X'E00

0E00 C8FF  INTRPT: ST       SAVA            ;SAVE ACCUM
0E02 40            LDE
0E03 C8FB          ST       SAVE            ;SAVE E REGISTER
0E05 06            CSA                      ;GET STATUS
0E06 C8F1          ST       SAVS            ;SAVE STATUS REGISTER
0E08 C455          LDI      L(MUXDIS)-1
0E0A 31            XPAL     1
0E0B C8F2          ST       SAV1LO
0E0D C40F          LDI      H(MUXDIS)
0E0F 35            XPAH     1
0E10 C8EC          ST       SAV1HI          ;PTR 1 SAVED
0E12 C480          LDI      L(RAM)
0E14 32            XPAL     2
0E15 C8E6          ST       SAV2LO
0E17 C40D          LDI      H(RAM)
0E19 36            XPAH     2
0E1A C8E0          ST       SAV2HI          ;OLD PTR 2 SAVED, NEW
                                            ;PTR2 = RAM
0E1C C429          LDI      L(SCAN)-1       ;GET KYBD SCAN ADDR
0E1E 33            XPAL     3
0E1F C8DA          ST       SAV3LO
0E21 C40F          LDI      H(SCAN)
0E23 37            XPAH     3
0E24 C8D4          ST       SAV3HI
0E26 C400          LDI      0
0E28 CA00          ST       TABLE(2)        ;CLEAR DISPLAY TABLE
0E2A CA01          ST       TABLE+1(2)
0E2C CA04          ST       TABLE+4(2)
```

Figure 2C2–13 (Continued)

```
110 0E2E CA05           ST      TABLE+5(2)
111 0E30 CA06           ST       STATUS(2)
112 0E32 CA0E           ST      INDX(2)
113 0E34 CA03           ST      TABLE+3(2)
114 0E36 C40D           LDI     X'D             ;'D'
115 0E38 CA02           ST      TABLE+2(2)
116 0E3A 3D             XPPC    1               ;DISPLAY '00D000'
117 0E3B 9039           JMP     CMLOOP
118 0E3D C400  EXIT:    LDI     0
119 0E3F CA06           ST      STATUS(2)       ;ZERO STATUS BITS
120 0E41 C0BC           LD      SAV1LO
121 0E43 31             XPAL    1
122 0E44 C0B8           LD      SAV1HI
123 0E46 35             XPAH    1               ;PTR 1 RESTORED
124 0E47 C0B4           LD      SAV2LO
125 0E49 32             XPAL    2
126 0E4A C0B0           LD      SAV2HI
127 0E4C 36             XPAH    2               ;PTR 2 RESTORED
128 0E4D C0AC           LD      SAV3LO
129 0E4F 33             XPAL    3
130 0E50 C0A8           LD      SAV3HI
131 0E52 37             XPAH    3               ;PTR 3 RESTORED
132 0E53 C0A4           LD      SAVS            ;GET OLD STATUS
133 0E55 07             CAS                     ;STATUS REG RESTORED
134 0E56 C0A8           LD      SAVE            ;GET OLD E REGISTER
135 0E58 01             XAE                     ;E REG RESTORED
136 0E59 C0A6           LD      SAVA            ;GET OLD ACCUM
137 0E5B 05             IEN                     ;ENABLE INTERRUPTS
138 0E5C 3F             XPPC    3               ;RETURN TO INTERRUPTED
139                                             ;PROGRAM
140 0E5D 90A1           JMP     INTRPT
141
142
143
144
145 0E5F C482  XECUTE:  LDI     L(G4HEX)-1
146 0E61 31             XPAL    1
147 0E62 C40F           LDI     H(G4HEX)
148 0E64 35             XPAH    1
149 0E65 3D             XPPC    1               ;GET 4 DIGIT HEX ADDR.
150 0E66 C208           LD      LADDR(2)        ;GET LO 8 ADDR
151 0E68 31             XPAL    1
152 0E69 C207           LD      HADDR(2)        ;GET HI 8 ADDR
153 0E6B 35             XPAH    1
154 0E6C C4FF           LDI     L(INTRPT)-1
155 0E6E 33             XPAL    3
156 0E6F C40E           LDI     H(INTRPT)
157 0E71 37             XPAH    3               ;RESTORE INTERRUPT POINTER
158 0E72 C5FF           LD      @-1(1)          ;DECREMENT PC PRIOR TO FETCH
159 0E74 05             IEN                     ;ENABLE INTERRUPT
160 0E75 3D             XPPC    1               ;EXECUTE USERS PROGRAM
161
162 0E76 3F    CMLOOP:  XPPC    3               ;CALL SCAN
163 0E77 40             LDE                     ;GET CODE
164 0E78 E413           XRI    X'13             ;MASK TO TEST  ABORT KEY
```

Figure 2C2–13 (Continued)

```
165 0E7A 98C1           JZ      EXIT            ;IF=1, EXIT INTRPT
166 0E7C 40             LDE
167 0E7D E410           XRI     X'10            ;MASK TO TEST  MODIFY COMMAND
168 0E7F 980C           JZ      MODIFY
169 0E81 40             LDE
170 0E82 E411           XRI     X'11            ;MASK TO TEST  XECUTE COMMAND
171 0E84 98D9           JZ      XECUTE
172 0E86 40             LDE
173 0E87 E412           XRI     X'12            ;MASK TO TEST  READ COMMAND
174 0E89 986B           JZ      READ
175 0E8B 90B0           JMP     EXIT            ;HEX ENTERED, ABORT
176
177             ;MODIFY COMMAND    ENTER COMMAND KEY FOLLOWED BY A 4 DIGIT
178                                ;HEX ADDRESS, THEN 2 HEX DIGITS OF DATA.
179                                ;AFTER FIRST ENTRY, THE OPERATOR CAN INCR
180                                ;AND LOAD SUCCESIVE LOCATIONS BY PRESSING
181                                ;MODIFY KEY FOLLOWED BY TWO HEX DATA ENTRIES
182
183
184 0E8D C206  MODIFY: LD      STATUS(2)       ;GET STATUS BIT
185 0E8F D401           ANI     1
186 0E91 9C2C           JNZ     UPDATE          ;IF 1, THIS KMODE
187                                             ;PREVIOUSLY ENTERED
188 0E93 C206           LD      STATUS(2)
189 0E95 D402           ANI     2
190 0E97 9806           JZ      MOD1            ;IF 0, MODIFY ENTERED
191                                             ;FOR FIRST TIME
192 0E99 C401           LDI     1
193 0E9B CA06           ST      STATUS(2)
194 0E9D 900B           JMP     DATA            ;READ WAS ENTERED
195 0E9F C401  MOD1:    LDI     1
196 0EA1 CA06           ST      STATUS(2)       ;SET MODE STATUS IF
197                                             ;THIS IS 1ST ENTRY
198 0EA3 C482           LDI     L(G4HEX)-1
199 0EA5 31             XPAL    1
200 0EA6 C40F           LDI     H(G4HEX)
201 0EA8 35             XPAH    1
202 0EA9 3D             XPPC    1               ;GET 4 HEX CHARACTERS
203 0EAA C48E  DATA:    LDI     L(G2HEX)-1
204 0EAC 31             XPAL    1
205 0EAD C40F           LDI     H(G2HEX)
206 0EAF 35             XPAH    1
207 0EB0 3D             XPPC    1
208 0EB1 C208           LD      LADDR(2)
209 0EB3 31             XPAL    1
210 0EB4 C207           LD      HADDR(2)
211 0EB6 35             XPAH    1               ;UPDATED ADDRESS RESTORED
212 0EB7 C209           LD      DATA1(2)        ;GET PACKED DIGITS
213 0EB9 C900           ST      (1)             ;LOCATION MODIFIED
214 0EBB 905C           JMP     G2DAT1
215
216
217             ;THIS SECTION OF CODE UPDATES THE ADDRESS POINTER
218             ;AND THE DISPLAY TABLE UPON SUCCESSIVE COMMAND
219             ;KEY ENTRIES
```

Figure 2C2–13 (Continued)

```
220
221
222 0EBD 90B7  LINK:   JMP     CMLOOP
223 0EBF 02     UPDATE: CCL
224 0EC0 C208           LD      LADDR(2)
225 0EC2 F401           ADI     1               ;INCREMENT LO ADDRESS
226 0EC4 CA08           ST      LADDR(2)
227 0EC6 06             CSA
228 0EC7 D480           ANI     X'80            ;MASK TO TEST  CARRY/LINK
229 0EC9 9807           JZ      PUTC
230 0ECB C207           LD      HADDR(2)
231 0ECD 02             CCL
232 0ECE F401           ADI     1               ;INCREMENT HI ADDRESS
233 0ED0 CA07           ST      HADDR(2)
234 0ED2 C208  PUTC:    LD      LADDR(2)
235 0ED4 D40F           ANI     X'F             ;BLANK UPPER 4
236 0ED6 CA03           ST      TABLE+3(2)      ;CHANGE ADDR DISPLAY LSD
237 0ED8 C208           LD      LADDR(2)
238 0EDA 1C             SR
239 0EDB 1C             SR
240 0EDC 1C             SR
241 0EDD 1C             SR
242 0EDE CA02           ST      TABLE+2(2)      ;CHANGE ADDR DISPLAY 2ND LSD
243 0EE0 C207           LD      HADDR(2)
244 0EE2 D40F           ANI     X'F
245 0EE4 CA01           ST      TABLE+1(2)      ;3RD LSD ADDR INTO
246                                             ;DISPLAY TABLE
247 0EE6 C207           LD      HADDR(2)
248 0EE8 1C             SR
249 0EE9 1C             SR
250 0EEA 1C             SR
251 0EEB 1C             SR
252 0EEC CA00           ST      TABLE(2)        ;MSD ADDR INTO
253                                             ;DISPLAY TABLE
254 0EEE C206           LD      STATUS(2)
255 0EF0 D402           ANI     2
256 0EF2 98B6           JZ      DATA
257 0EF4 901D           JMP     G2DATA          ;GET 2 HEX DATA & DISPLAY
258
259
260
261
262 0EF6 C206  READ:    LD      STATUS(2)       ;GET PROGRAM STATUS WORD
263 0EF8 D402           ANI     2
264 0EFA 9CC3           JNZ     UPDATE          ;IF 1, THIS KMODE
265                                             ;PREVIOUSLY ENTERED
266 0EFC C206           LD      STATUS(2)
267 0EFE D401           ANI     1               ;MASK TO TEST  MODIFY
268 0F00 9806           JZ      READ1           ;IF 0, READ ENTERED FOR FIRST
269                                             ;TIME
270 0F02 C402           LDI     2
271 0F04 CA06           ST      STATUS(2)
272 0F06 900B           JMP     G2DATA          ;MODIFY WAS ENTERED
273 0F08 C402  READ1:   LDI     2
274 0F0A CA06           ST      STATUS(2)       ;SET MODE STATUS WORD
```

Figure 2C2–13 (Continued)

```
275 0F0C C482           LDI     L(G4HEX)-1
276 0F0E 31             XPAL    1
277 0F0F C40F           LDI     H(G4HEX)
278 0F11 35             XPAH    1
279 0F12 3D             XPPC    1               ;GET 4 HEX DIGIT ADDRESS
280 0F13 C208  G2DATA:  LD      LADDR(2)
281 0F15 31             XPAL    1
282 0F16 C207           LD      HADDR(2)
283 0F18 35             XPAH    1               ;PTR 1 = ADDR TO BE READ
284 0F19 C100  G2DAT1:  LD      (1)             ;GET DATA
285 0F1B 01             XAE                     ;SAVE DATA IN E REG
286 0F1C 40             LDE                     ;GET PACKED DATA
287 0F1D 1C             SR
288 0F1E 1C             SR
289 0F1F 1C             SR
290 0F20 1C             SR                      ;MSD IN LO 4
291 0F21 CA04           ST      TABLE+4(2)      ;MSD DATA INTO TABLE
292 0F23 40             LDE                     ;GET DATA
293 0F24 D40F           ANI     X'F             ;BLANK UPPER 4
294 0F26 CA05           ST      TABLE+5(2)      ;LSD DATA INTO TABLE
295 0F28 9093           JMP     LINK
296
297
298
299
300
301             ;NEXT SECTION OF CODE IS THE KEYBOARD SCAN AND JUMP TO
302             ;THE MULTIPLEXED DISPLAY ROUTINE
303
304
305 0F2A C455  SCAN:    LDI     L(MUXDIS)-1
306 0F2C 31             XPAL    1
307 0F2D CA0A           ST      LTEMP2(2)
308 0F2F C40F           LDI     H(MUXDIS)
309 0F31 35             XPAH    1
310 0F32 CA0B           ST      HTEMP2(2)
311 0F34 06    LOOK:    CSA
312 0F35 D410           ANI     X'10            ;TEST SENSE A
313 0F37 9C03           JNZ     INPUT           ;IF 1, KEY IS PRESSED
314 0F39 3D             XPPC    1               ;REFRESH THE DISPLAY
315 0F3A 90F8           JMP     LOOK
316 0F3C C2FF  INPUT:   LD      KYBD(2)         ;GET KEYCODE
317 0F3E D41F           ANI     X'1F            ;BLANK UPPER 5
318 0F40 CA0F           ST      ESAV(2)         ;SAVE CODE
319 0F42 06    RELEAS:  CSA
320 0F43 D410           ANI     X'10
321 0F45 9803           JZ      RETURN          ;IF 0, KEY RELEASED
322 0F47 3D             XPPC    1               ;REFRESH THE DISPLAY
323 0F48 90F8           JMP     RELEAS
324 0F4A C20A  RETURN:  LD      LTEMP2(2)
325 0F4C 31             XPAL    1
326 0F4D C20B           LD      HTEMP2(2)
327 0F4F 35             XPAH    1               ;PTR 1 RESTORED
328 0F50 C20F           LD      ESAV(2)         ;GET CODE
329 0F52 01             XAE
```

Figure 2C2–13 (Continued)

```
330 0F53 3F            XPPC    3               ;RETURN TO CALLING ROUTINE
331 0F54 90D4          JMP     SCAN
332
333
334             ;THE NEXT SECTION OF CODE PERFORMS THE MULTIPLEX
335             ;REFRESH OF THE DISPLAY.
336
337
338             ;THE 6 DIGITS TO BE DISPLAYED ARE STORED IN 6
339             ;CONSECUTIVE LOCATIONS STARTING AT X'D80.
340
341
342 0F56 C400  MUXDIS: LDI     L(LEDS)
343 0F58 31            XPAL    1
344 0F59 CA0C          ST      SAV1(2)         ;SAVE OLD P1 LO
345 0F5B C40D          LDI     H(LEDS)
346 0F5D 35            XPAH    1
347 0F5E CA0D          ST      SAV2(2)         ;SAVE OLD P1 HI
348
349             ;POINTER 1 NOW IS THE BASE ADDRESS OF THE DISPLAY
350
351
352
353 0F60 C420           LDI     X'20
354 0F62 01    LOOP:   XAE                     ;E REG = DIGIT DRIVE
355 0F63 C601          LD      @1(2)           ;GET HEX DIGIT
356 0F65 C980          ST      -128(1)         ;DISPLAY THE DIGIT
357 0F67 40            LDE                     ;GET DIGIT DRIVE
358 0F68 1C            SR                      ;CHANGE DIGIT
359 0F69 9CF7          JNZ     LOOP
360 0F6B C900          ST      (1)             ;BLANK DISPLAY
361 0F6D C480          LDI     L(RAM)
362 0F6F 32            XPAL    2               ;RESTORE PTR 2 LO
363 0F70 C20D          LD      SAV2(2)         ;GET OLD PTR 1 HI
364 0F72 35            XPAH    1
365 0F73 C20C          LD      SAV1(2)         ;GET OLD PTR 1 LO
366 0F75 31            XPAL    1
367 0F76 3D            XPPC    1               ;RETURN TO CALLING PROG
368 0F77 90DD          JMP     MUXDIS
369
370
371
372
373             ;THE G4HEX ROUTINE GETS 4 HEX DIGITS AND STORES THEM
374             ;IN THE DISPLAY TABLE.  IF A COMMAND KEY IS ENTERED
375             ;DURING THIS SEQUENCE, THE INTERRUPT SERVICE IS EXITED
376             ;DISPLAY TABLE IS AT X'D80 THRU X'D85
377
378
379 0F79 C480  ERROR:  LDI     L(RAM)
380 0F7B 32            XPAL    2
381 0F7C C43C          LDI     L(EXIT)-1
382 0F7E 31            XPAL    1
383 0F7F C40E          LDI     H(EXIT)
384 0F81 35            XPAH    1
```

Figure 2C2–13 (Continued)

```
385 0F82 3D             XPPC    1               ;EXIT INTERUPT
386 0F83 31    G4HEX:   XPAL    1
387 0F84 CA13           ST      SAVL1(2)
388 0F86 35             XPAH    1
389 0F87 CA14           ST      SAVL2(2)
390 0F89 3F             XPPC    3               ;CALL SCAN
391 0F8A C404           LDI     4
392 0F8C 31             XPAL    1
393 0F8D 9014           JMP     FIRST
394 0F8F 31    G2HEX:   XPAL    1               ;SAVE P1 LO
395 0F90 CA13           ST      SAVL1(2)        ;SAVE P1 LO
396 0F92 35             XPAH    1
397 0F93 CA14           ST      SAVL2(2)
398 0F95 3F             XPPC    3               ;CALL SCAN. RETURN WITH CODE
399                                             ;IN E REG.
400 0F96 C402           LDI     2
401 0F98 31             XPAL    1
402 0F99 C604           LD      @4(2)           ;MODIFY P2 FOR INDEXING
403 0F9B 9006           JMP     FIRST
404 0F9D C212  LOOP1:   LD      CNT(2)
405 0F9F 31             XPAL    1
406 0FA0 C20E           LD      INDX(2)         ;GET PREVIOUS INDEX VALUE
407 0FA2 32             XPAL    2               ;SET UP P2
408
409
410             ;PTR 2 IS AUTO INDEXED THRU THE DISPLAY TABLE.
411             ;THE 1ST FOUR LOCATIONS CONTAIN THE HEX ADDRESS
412             ;TO BE DISPLAYED.
413
414
415 0FA3 40    FIRST:  LDE.                    ;GET CODE
416 0FA4 D410           ANI     X'10            ;MASK TO TEST  COMMAND
417 0FA6 9CD1           JNZ     ERROR           ;IF 1, ABORT INTRPT
418 0FA8 40             LDE
419 0FA9 D40F           ANI     X'F
420 0FAB CE01           ST      @1(2)
421 0FAD 31             XPAL    1               ;GET LOOP COUNT
422 0FAE 02             CCL
423 0FAF F4FF           ADI     -1              ;DECR LOOP COUNT
424 0FB1 980C           JZ      DONE
425 0FB3 31             XPAL    1               ;REPLACE LOOP COUNT
426 0FB4 C480           LDI     L(RAM)
427 0FB6 32             XPAL    2               ;RESTORE P2
428 0FB7 CA0E           ST      INDX(2)         ;SAVE INDEX VALUE
429 0FB9 31             XPAL    1
430 0FBA CA12           ST      CNT(2)
431 0FBC 3F             XPPC    3               ;CALL SCAN
432 0FBD 90DE           JMP     LOOP1
433
434
435             ;THE SECTION OF CODE BEGINNING WITH 'DONE', TAKES
436             ;ADDRESS AND DATA FROM THE DISPLAY TABLE LOCATIONS
437             ;AND PACKS INTO LOCATIONS LADDR, HADDR, AND DATA1.
438
439
```

Figure 2C2–13 (Continued)

```
440
441
442 0FBF C480  DONE:   LDI     L(RAM)
443 0FC1 32             XPAL    2               ;RESTORE P2
444 0FC2 C403           LDI     3
445 0FC4 31             XPAL    1               ;P1 IS LOOP CNTR
446 0FC5 C487           LDI     X'87            ;NEW BASE FOR P3
447 0FC7 33             XPAL    3               ;MODIFY P3 FOR INDEXING
448 0FC8 CA10           ST      LO3(2)          ;SAVE P3 LO
449 0FCA C40D           LDI     H(RAM)
450 0FCC 37             XPAH    3
451 0FCD CA11           ST      HI3(2)
452 0FCF C601  PACK:    LD      @1(2)           ;GET CODE FROM TABLE
453 0FD1 1E             RR
454 0FD2 1E             RR
455 0FD3 1E             RR
456 0FD4 1E             RR                      ;SHIFTED LEFT 4
457 0FD5 01             XAE
458 0FD6 C601           LD      @1(2)
459 0FD8 58             ORE
460 0FD9 CF01           ST      @1(3)           ;HADDR/LADDR/DATA1
461 0FDB 31             XPAL    1               ;GET LOOP COUNT
462 0FDC 02             CCL
463 0FDD F4FF           ADI     -1              ;DECR
464 0FDF 9803           JZ      OUT
465 0FE1 31             XPAL    1               ;RESTORE COUNT
466 0FE2 90EB           JMP     PACK
467 0FE4 C480  OUT:     LDI     L(RAM)
468 0FE6 32             XPAL    2               ;RESTORE P2
469 0FE7 C210           LD      LO3(2)
470 0FE9 33             XPAL    3
471 0FEA C211           LD      HI3(2)
472 0FEC 37             XPAH    3               ;RESTORE P3
473 0FED C213           LD      SAVL1(2)
474 0FEF 31             XPAL    1               ;RESTORE P1
475 0FF0 C214           LD      SAVL2(2)
476 0FF2 35             XPAH    1
477 0FF3 3D             XPPC    1               ;RETURN TO CALLING COMMAND
478
479       0000           END
```

CMLOOP	0E76	CNT	0012	DATA	0EAA
DATA1	0009	DONE	0FBF	ERROR	0F79
ESAV	000F	EXIT	0E3D	FIRST	0FA3
G2DAT1	0F19	G2DATA	0F13	G2HEX	0F8F
G4HEX	0F83	HADDR	0007	HI3	0011
HTEMP2	000B	INDX	000E	INPUT	0F3C
INTRPT	0E00	KYBD	FFFF	LADDR	0008
LEDS	0D00	LINK	0EBD	LO3	0010
LOOK	0F34	LOOP	0F62	LOOP1	0F9D
LTEMP2	000A	MOD1	0E9F	MODIFY	0E8D
MUXDIS	0F56	OUT	0FE4	PACK	0FCF
PUTC	0ED2	RAM	0D80	READ	0EF6
READ1	0F08	RELEAS	0F42	RETURN	0F4A
SAV1	000C	SAV1HI	0DFD	SAV1LO	0DFE

Figure 2C2–13 (Continued)

```
SAV2     000D          SAV2HI   0DFB          SAV2LO   0DFC
SAV3HI   0DF9          SAV3LO   0DFA          SAVA     0E00
SAVE     0DFF          SAVL1    0013          SAVL2    0014
SAVS     0DF8          SCAN     0F2A          STATUS   0006
TABLE    0000          UPDATE   0EBF          XECUTE   0E5F


NO ERROR LINES
SOURCE CHECKSUM=1050
```

NS10555

Figure 2C2–13 (Concluded)

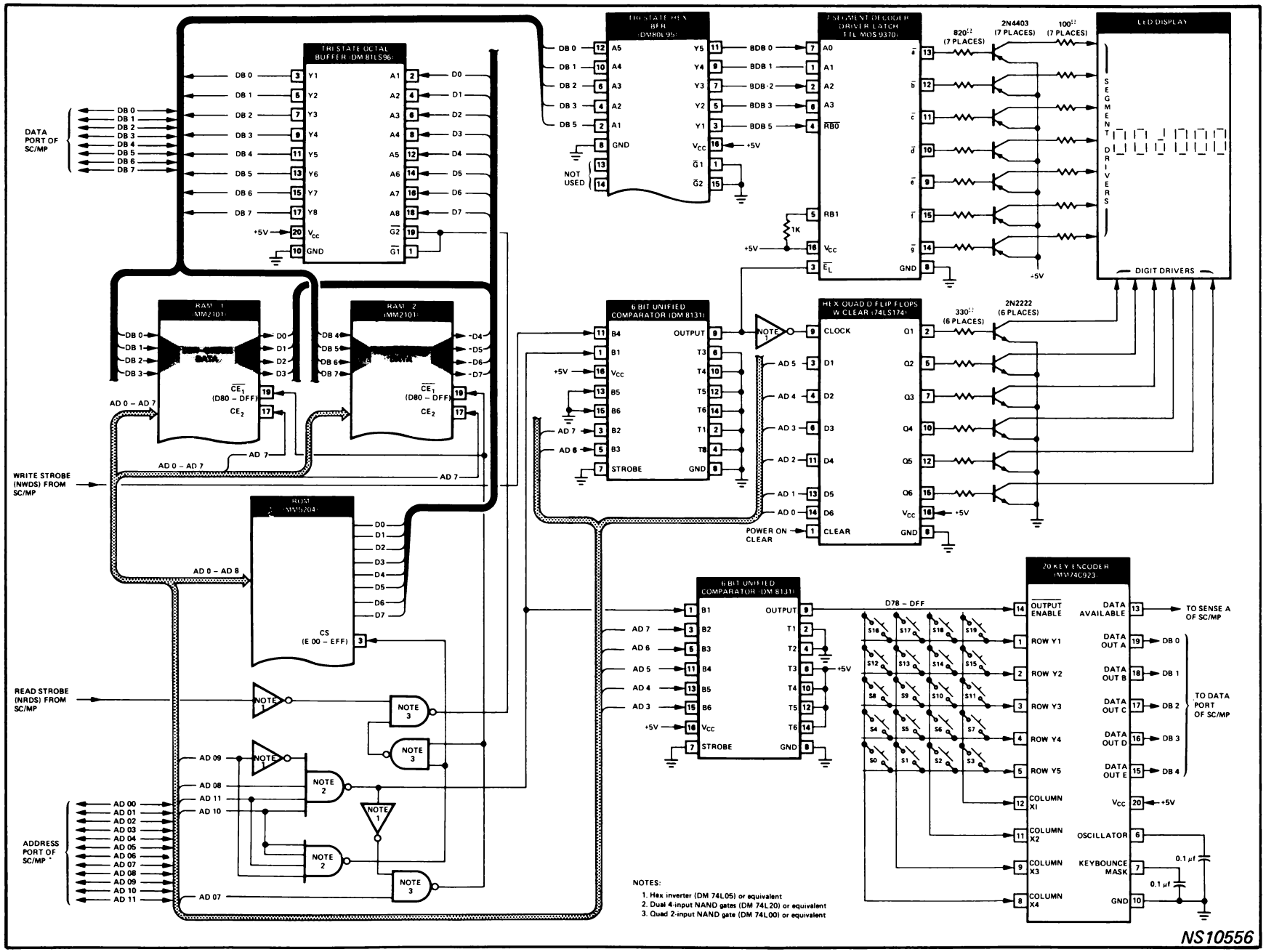


Figure 2C2–14. Interrupt Driven Keyboard/Display System–Schematic Diagram

INTERFACING SC/MP WITH A BURROUGHS SELF-SCAN DISPLAY

General Description

The SC/MP-and-display interface shown in figure 2C2-15 provides an easy and relatively inexpensive method of generating 64 alphanumeric characters – any 32 of which are simultaneously shown on a single-row display panel. This SC/MP-based system can be used efficiently for any close-view (up to 10 feet) moving-message or static display.

System Operation

As shown in figure 2C2-15, data are input from SC/MP via a low-power TRI-STATE buffer (DM81LS95) and these data are latched by the DM74199. Under software supervision, flag 0 is used to generate the "write" pulse and Sense B is used to indicate "status" – the status specifying when a new character can be input to the display. Each of the 64 characters is defined by a 6-bit binary-to-hexadecimal code; the characters and their equivalent hexadecimal codes are shown in table 2C2-1.

Table 2C2-1. Alphanumeric Characters and Corresponding Hex-Input Codes

HEX INPUT	CHARACTER	HEX INPUT	CHARACTER	HEX INPUT	CHARACTER	HEX INPUT	CHARACTER
00	@	10	P	20	(BLANK)	30	0
01	A	11	Q	21	!	31	1
02	B	12	R	22	"	32	2
03	C	13	S	23	#	33	3
04	D	14	T	24	$	34	4
05	E	15	U	25	·/.	35	5
06	F	16	V	26	&	36	6
07	G	17	W	27	/	37	7
08	H	18	X	28	<	38	8
09	I	19	Y	29	>	39	9
A	J	1A	Z	2A	*	3A	:
B	K	1B	[	2B	+	3B	;
C	L	1C	~	2C	'	3C	<
D	M	1D	]	2D	–	3D	=
E	N	1E	{	2E	•	3E	>
F	O	1F	}	2F	⌐⌋	3F	?

Software Considerations

Memory interfaces for the SC/MP-display system are shown in figure 2C2-15. The control program is stored in ROM – X'000 through X'01FF; RAM utilizes locations X'0F00 through X'0FFF with a display address of X'0800. There are no special timing restraints required to communicate with the self-scan display.

Each character of the message is brought in from the buffer; then, the program checks to see if the character is valid, and if it is valid, the software converts the 7-bit ASCII input code to a 6-bit ASCII output code. After this conversion is made, the clear bit and display-blanking bit are set to the proper condition and are ORed with the character. The character word now is written into the DM74199 latch. Subsequently, the Data Present line is pulsed and the Write Flag is tested to see if the display is ready to accept new data. Figures 2C2-16 and 2C2-17, respectively, show the flowchart and the program listing for the Control and Message-Moving Program that is used to print a message that is greater than 32 words long.

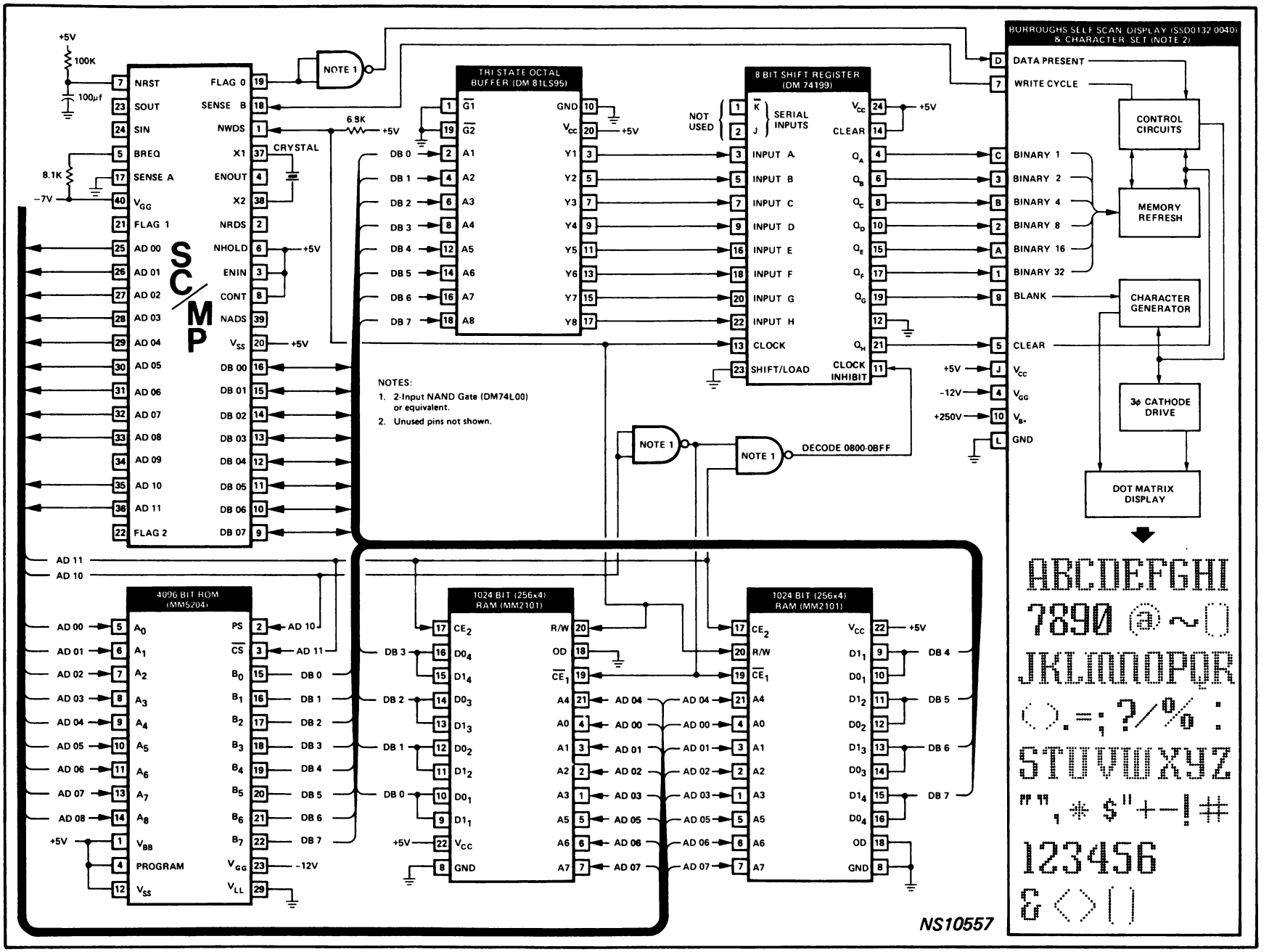


Figure 2C2–15. SC/MP Interfaced with Burroughs Self-Scan Display

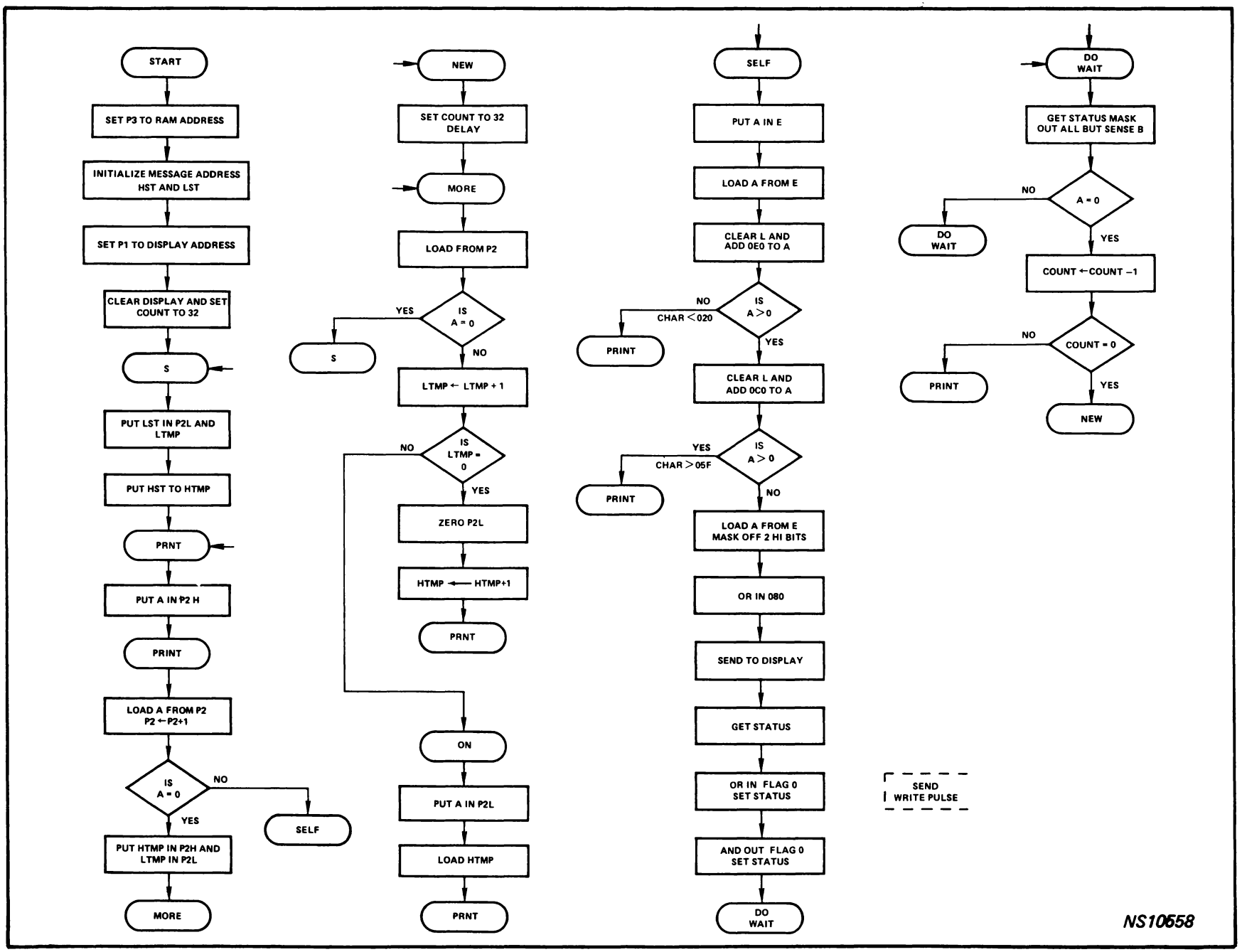


Figure 2C2–16. Flowchart for Control and Moving-Message Program

```
                        TITLE  DISP. 'MOVING MESSAGE

                       MESSAGE MUST BE   32 CHARACTERS.


                       3/17/76


                       RAM LOCATIONS USED.


                       0F00    ADDRESS OF MESSAGE HIGH
                       0F01    ADDRESS OF MESSAGE LOW
                       0F02    NEXT ADDRESS OF MESSAGE HIGH
                       0F03    NEXT ADDRESS OF MESSAGE LOW
                       0F04    CHAR PER LINE COUNTER


      0000  MSI      =        0                ;SAME AS 0F00
      0001  LSI      =        1                ;SAME AS 0F01
      0002  HTMP     =        2                ;SAME AS 0F02
      0003  LTMP     =        3                ;SAME AS 0F03
      0004  COUNT              4                ;SAME AS 0F04
      0800  ADR      =        0800             ;ADDRESS OF DISPLAY.
      0F00  RAM      =        0F00             ;START OF RAM.

      0000                    0                ;STARTING ADDRESS.


                       MESSAGE IS ASCII STRING IN MEMORY.
                       END OF MESSAGE IS A BYTE OF ALL 0.


      0F20  MMSG               0F20             ;ADDRESS OF MESSAGE.


                        PAGE

              START:
0000 0000 08            NOP
0001 C40F               LDI     H(RAM)           ;PUT RAM ADDRESS IN P3.
0003 37                 XPAH    3
0004 C400               LDI     L(RAM)
0006 33                 XPAL    3
0007 C40F               LDI     H(MMSG)          ;SET STARTING ADDRESS IF MESSA
0009 CB00               ST      MSI(3)           ;SAVE IN RAM.
000B C420               LDI     L(MMSG)
000D C901               ST      LSI(3)
000F C408               LDI     H(ADR)           ;PUT ADDRESS OF DISPLAY IN P1.
0011 35                 XPAH    1
0012 C400               LDI     L(ADR)
0014 31                 XPAL    1
0015 C400               LDI     0                ;CLEAR DISPLAY.
```

Figure 2C2–17. Program Listing for Control and Moving-Message Program

```
 56 0017 C900           ST      (1)
 57 0019 C420           LDI     32              ;SET LINE COUNT.
 58 001B CB04           ST      COUNT(3)
 59                 S.
 60 001D C301           LD      LST(3)          ;PUT ADDRESS IN TEMP.
 61 001F CB03           ST      LTMP(3)
 62 0021 32             XPAL    2               ;SET P2 TO ADDRESS.
 63 0022 C300           LD      HST(3)
 64 0024 CB02           ST      HTMP(3)
 65                 PRNT:
 66 0026 36             XPAH    2               ;HIGH ADDRESS IN P2.
 67                 PRINT:
 68 0027 C601           LD      @1(2)           ;GET NEXT WORD.
 69 0029 9C22           JNZ     SELF            ;CHECK IF DONE.
 70 002B C302           LD      HTMP(3)         ;RESTORE POINTER.
 71 002D 36             XPAH    2
 72 002E C303           LD      LTMP(3)
 73 0030 32             XPAL    2
 74 0031 90DD           JMP     MORE
 75                 ON:
 76 0033 32             XPAL    2               ;SAVE IN P2 LOW.
 77 0034 C302           LD      HTMP(3)         ;RESTORE HIGH.
 78 0036 90EE           JMP     PRNT
 79                 NEW:
 80 0038 C420           LDI     32              ;SAVE LINE COUNT.
 81 003A CB04           ST      COUNT(3)
 82 003C C4FF           LDI     0FF             ;DO A SHORT DELAY.
 83 003E 8F00           DLY     000
 84                 MORE:
 85 0040 C200           LD      (2)             ;CHECK IF DONE.
 86 0042 98D9           JZ      S
 87 0044 AB03           ILD     LTMP(3)         ;BUMP RAM POINTER.
 88 0046 9CEB           JNZ     ON
 89 0048 32             XPAL    2               ;NEXT ADDRESS.
 90 0049 AB02           ILD     HTMP(3)         ;BUMP HIGH.
 91 004B 90DA           JMP     PRINT
 92                 SELF:
 93 004D 01             XAE                     ;SAVE CHAR.
 94 004E 40             LDE                     ;GET CHAR.
 95 004F 02             CCL                     ;CLEAR LINK.
 96 0050 F420           ADI     0E0             ;CHECK IF LESS THAN 020.
 97 0052 9401           JP      GTIF            ;NO > 01F.
 98 0054 90D1           JMP     PRINT           ;LESS THAN 01F RETURN.
 99                 GTIF
100 0056 02             CCL                     ;CLEAR LINK.
101 0057 F400           ADI     000             ;CHECK IF > 05F.
102 0059 94CC           JP      PRINT           ;YES RETURN.
103 005B 40             LDE                     ;CHAR IS VALID.
104 005C D43F           ANI     03F             ;STRIP OFF HIGH BITS.
105 005E DC00           ORI     000             ;SET CLEAR AND DISPLAY BITS.
106 0060 C900           ST      (1)             ;SEND WORD.
107 0062 06             LSA
108 0063 DC01           ORI     1               ;SET WRITE CYCLE FLAG 0.
109 0065 07             LAS                     ;NOW SET FLAG 0.
110 0066 D4FE           ANI     0FE             ;NOW RESET FLAG.
111 0068 07             LAS                     ;DO IT.
```

Figure 2C2–17 (Continued)

```
112                DOWAIT:
113  0069 06               CSA                         ;GET STATUS.
114  006A D420             ANI       020               ;CHECK IF SENSE B IS SET.
115  006C 9CF8             JNZ       DOWAIT            ;WAIT IF SET.
116  006E B604             DLD       COUNT(3)          ;BUMP COUNTER.
117  0070 9806             JZ        NEW
118  0072 9803             JMP       PRINT
119
120
121
122        0000              END                        •

          ADR       0800           COUNT     0004           DOWAIT    0069
          GT1F      0056           HST       0000           HTMP      0002
          LST       0001           LTMP      0003           MMSG      0F20
          MORE      0040           NEW       0038           ON        0033
          PRINT     0027           PRNT      0026           RAM       0F00
          S         001D           SELF      004D           START     0000 *

          NO ERROR LINES
          SOURCE CHECKSUM=39A5
```

NS10559

Figure 2C2–17 (Concluded)

MULTIPROCESSOR SYSTEM

General Description

Figure 2C3-1 shows how two microprocessors –SC/MP #1 and SC/MP #2 – can be interconnected to perform different tasks on a time- and memory-share basis. SC/MP #1 is the basic SC/MP kit (1SP-8K/200) with a TTY input/output interface, whereas SC/MP #2 is used to drive a Burroughs self-scan display – see figure 2C2-15. The control program for each microprocessor is stored in a separate ROM; the 256 bytes of RAM are shared. The basic functions of SC/MP #1 are defined in the SC/MP Kit Users Manual; however, with more RAM and with the latching and buffering techniques shown in figure 1-8, the kit capabilities can be expanded to provide a complete keyboard/display system.

System Operation

When power is applied to the multiprocessor system, NRST of SC/MP #2 is driven high via the RC network and the processor initializes at address X'0001; for the time being, SC/MP #1 is held off by the low input from flag 1 of SC/MP #2. Each processor uses the Sense A input for program direction; that is, if Sense A is low for SC/MP #2, it branches to the Burroughs self-scan routine. After this decision is implemented by setting P3 to the proper address, the SC/MP #2 software sets flag 1 high (NRST #1 now is driven high); this causes SC/MP #1 to initialize at address X'0001. Since the Sense A input of SC/MP #1 is tied high, it branches to the KITBUG Routine.

With both processors initialized and directed to their respective programs, bus requests are made and ENIN is tested for bus access. If ENIN is low, the processor requesting access must wait until its ENIN lines goes high. If the ENIN line is high and no bus request is issued, the "bus available" signal is passed to the next processor – in this case, SC/MP #2 – via the ENOUT line. (Refer to figure 1-4f for functional detail of bus-access control.)

Software Considerations

Figure 2C3-2 shows the system flowchart and the program listing for KITBUG; a detailed flowchart for KITBUG is shown in the SC/MP Kit Users Manual. The program listing for the self-scan routine is shown in figure 2C3-3; except for address assignments, this listing is similar to that shown in figure 2C2-17. The flowchart for the multiprocessor "self-scan" program is functionally equivalent to that shown in figure 2C2-16. There are no timing constraints; the system is self-clocking and self-synchronizing.

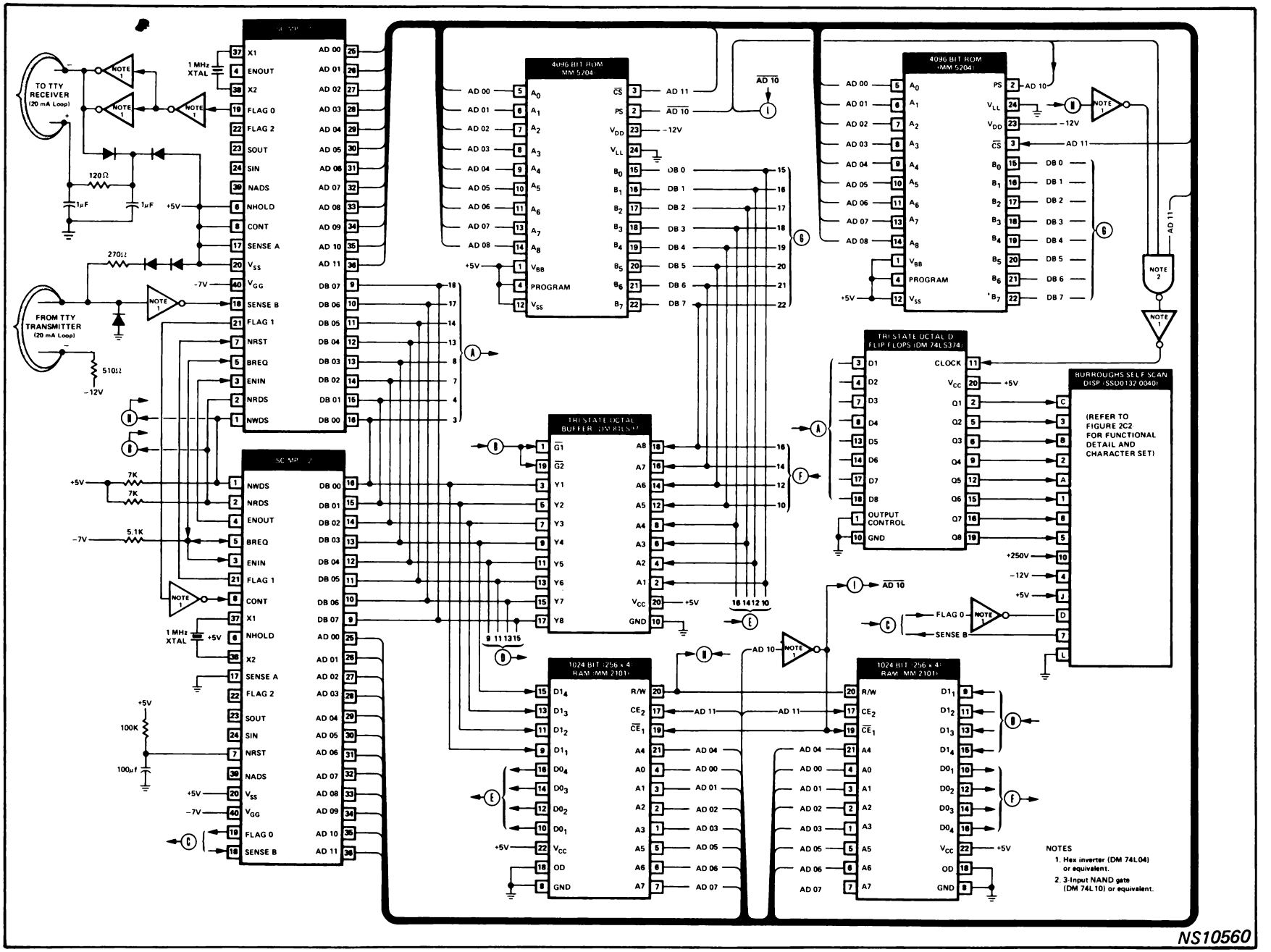


Figure 2C3–1. Using SC/MP in a Multiprocessing System

```
 1                          .TITLE  KITBUG,' MULTI PROCESSOS 3 17 76 '
 2
 3
 4        0001  P1          =       1
 5        0002  P2          =       2
 6        0003  P3          =       3
 7
 8        FFFF  EXOFF       =       -1
 9
10              ; FIXED STACK ASSIGNMENTS
11
12        0FFF              .=0FFF
13              STACK:
14        0000  SR          =       .-STACK
15        0FFE              .=.-1
16        FFFF  EX          =       .-STACK
17        0FFD              .=.-1
18        FFFE  AC          =       .-STACK
19        0FFB              .=.-2
20        FFFC  PT2         =       .-STACK
21        0FF9              .=.-2
22        FFFA  PT1         =       .-STACK
23        0FF7              .=.-2
24        FFF8  PC          =       .-STACK
25
26        0FF6  P2ADR       =       .-1
27
28
29
30
31        0000              .=0
32  0000  08                NOP
33              START:
34  0001  06                CSA                     ;READ STATUS
35  0002  D410              ANI     010             ;CHECK FOR SENSE A.
36  0004  9C22              JNZ     ENTER           ;SET/RESET DISPLAY/KITBUG.
37  0006  33                XPAL    P3              ;ZERO P3 LOW.
38  0007  C404              LDI     004             ;SET ADDRESS OF ROM2.
39  0009  37                XPAH    P3
40  000A  3F                XPPC    P3              ;GO TO DISPLAY.
41
42
                            .PAGE   'DEBUG ENTRY AND EXIT'
43                          .LOCAL
44
45              ; ON A SOFTWARE HALT, HARDWARE USES THE FOLLOWING WORDS
46              ; TO SAVE THE ENVIROMENT.
47
48              ; DEBUG EXIT - RESTORE ENVIROMENT AND GO.
49
50  000B  C0F2  EXIT:   LD      STACK+EX        ; RESTORE E REG
51  000D  01            XAE
```

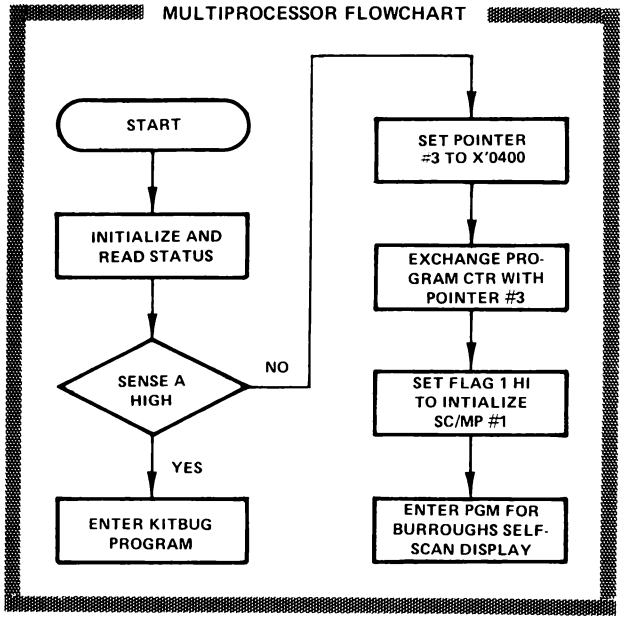


Figure 2C3–2. Flowchart for Multiprocessor System and Program Listing for KITBUG

```
 52 000E C0EA           LD      STACK+PT1       ; RESTORE P1
 53 0010 35             XPAH    P1
 54 0011 C0E8           LD      STACK+PT1+1
 55 0013 31             XPAL    P1
 56 0014 C0E6           LD      STACK+PT2       ; RESTORE P2
 57 0016 36             XPAH    P2
 58 0017 C0E4           LD      STACK+PT2+1
 59 0019 32             XPAL    P2
 60 001A C0DC           LD      STACK+PC        ; PUT DESIRED PC IN P3
 61 001C 37             XPAH    P3
 62 001D C0DA           LD      STACK+PC+1
 63 001F 33             XPAL    P3
 64 0020 C7FF           LD      @EXOFF(P3)      ; ADD EXIT OFFSET TO PC
 65 0022 C0DC           LD      STACK+SR        ; RESTORE SR
 66 0024 07             CAS
 67 0025 C0D7           LD      STACK+AC
 68 0027 3F             XPPC    P3
 69
 70             ; DEBUG ENTRY POINT
 71
 72 0028 C8D4  ENTER:   ST      STACK+AC
 73 002A 06             CSA
 74 002B C8D3           ST      STACK+SR
 75 002D 01             XAE                     ; SAVE EXTENSION REGISTER
 76 002E C8CF           ST      STACK+EX
 77 0030 36             XPAH    P2              ;   POINTER
 78 0031 C8C9           ST      STACK+PT2
 79 0033 32             XPAL    P2
 80 0034 C8C7           ST      STACK+PT2+1
 81 0036 35             XPAH    P1              ;   STACK
 82 0037 C8C1           ST      STACK+PT1
 83 0039 31             XPAL    P1
 84 003A C8BF           ST      STACK+PT1+1
 85 003C 37             XPAH    P3
 86 003D C8B9           ST      STACK+PC
 87 003F 33             XPAL    P3
 88 0040 C8B7           ST      STACK+PC+1
 89
                        .PAGE   'MAIN COMMAND LOOP'
 90                     .LOCAL
 91
 92             ; THIS CODE INITIALIZES POINTER REGISTERS AND
 93             ; PROMPTS FOR AND GETS THE NEXT COMMAND.
 94
 95             ; ON EXIT, E HOLDS THE COMMAND CHARACTER
 96
 97 0042 C4F6  CMDLP:   LDI     L(P2ADR)
 98 0044 32             XPAL    P2
 99 0045 C40F           LDI     H(P2ADR)
100 0047 36             XPAH    P2
101 0048 C401           LDI     H(PUTC)         ; PRINT CR-LF
102 004A 37             XPAH    P3
103 004B C4C3           LDI     L(PUTC)-1
```

Figure 2C3–2 (Continued)

```
104 004D 33             XPAL    P3
105 004E C40D           LDI     0D
106 0050 3F             XPPC    P3              ; PRINT CR
107 0051 C40A           LDI     0A
108 0053 3F             XPPC    P3              ; PRINT LF
109 0054 C42D           LDI     '-'
110 0056 3F             XPPC    P3
111 0057 C484           LDI     L(GECO)-1       ;P3 HIGH OK.
112 0059 33             XPAL    P3              ; GET COMMAND CHAR.
113 005A 3F             XPPC    P3
114
                        .PAGE   'GO'
115                     .LOCAL
116
117             ; RESTORE MACHINE STATE AND TRANSFER CONTROL
118             ; TO SPECIFIED ADDRESS.
119
120             ; G ADDRESS
121
122 005B 40     GO:     LDE
123 005C E447           XRI     'G'
124 005E 9C07           JNZ     $SKIP
125 0060 3F             XPPC    P3              ; CALL GECO
126 0061 E40D           XRI     0D
127 0063 98A6           JZ      EXIT
128 0065 9064           JMP     ERROR
129             $SKIP:
130
                        .PAGE   'TYPE'
131                     .LOCAL
132
133             ; TYPE OR MODIFY MEMORY.
134
135 0067 40     TYPE:   LDE                     ; CHECK FOR TYPE COMMAND, IF
136 0068 E454           XRI     'T'             ;   NOT 'T', SKIP COMMAND.
137 006A 9809           JZ      $2
138 006C 40     MOD:    LDE
139 006D E44D           XRI     'M'
140 006F 9C5A           JNZ     $SKIP
141 0071 C400           LDI     0
142 0073 9002           JMP     $1
143 0075 C401   $2:     LDI     1
144 0077 CEFF   $1:     ST      @-1(P2)         ; SAVE FLAG FOR TYPE OR MODIFY
145 0079 C400           JS      P3,GHEX         ; GET ADDRESS
    007B 37C4
    007D DE33
    007F 3F
146 0080 E40D           XRI     0D              ; CHECK TERMINATOR
147 0082 9C47           JNZ     ERROR
148 0084 C601           LD      @1(P2)          ; PUT STARTING ADDRESS IN STAC
149 0086 35             XPAH    P1
150 0087 C601           LD      @1(P2)
151 0089 31             XPAL    P1
152 008A C401   $4:     LDI     H(PUTC)         ; PRINT CR-LF
```

Figure 2C3–2 (Continued)

```
153 008C 37             XPAH    P3
154 008D C4C3           LDI     L(PUTC)-1
155 008F 33             XPAL    P3
156 0090 C40D           LDI     0D
157 0092 3F             XPPC    P3              ; PRINT CR
158 0093 C40A           LDI     0A
159 0095 3F             XPPC    P3              ; PRINT LF
160 0096 35             XPAH    P1              ; PRINT HIGH BYTE
161 0097 01             XAE                     ; READ AND RESTORE BYTE FROM P
162 0098 40             LDE
163 0099 35             XPAH    P1
164                                             ;P3 HIGH OK.
165 009A C442           LDI     L(PHEX2)-1
166 009C 33             XPAL    P3
167 009D 40             LDE
168 009E 3F             XPPC    P3              ; CALL PHEX2
169 009F 31             XPAL    P1              ; PRINT LOW BYTE
170 00A0 01             XAE
171 00A1 40             LDE
172 00A2 31             XPAL    P1
173 00A3 40             LDE
174 00A4 3F             XPPC    P3              ; CALL PHEX1
175 00A5 C501           LD      @1(P1)
176 00A7 3F             XPPC    P3              ; PRINT 2-DIGIT HEX FOLLOWED B
177                                             ;   BLANK  (PHEX1)
178 00A8 C200           LD      (P2)            ; CHECK TYPE OR MODIFY FLAG
179 00AA 9CDE           JNZ     $4
180                                             ;P3 HIGIH IS STILL OK.
181 00AC C484           LDI     L(GECO)-1
182 00AE 33             XPAL    P3
183 00AF 3F             XPPC    P3              ;GO TO GECO.
184 00B0 E40D           XRI     0D
185 00B2 98D6           JZ      $4
186 00B4 E415           XRI     015             ; 0D XOR 018 (CAN)
187 00B6 988A  LOOP1:   JZ      CMDLP
188 00B8 C400           JS      P3,GHEX2
    00BA 37C4
    00BC DA33
    00BE 3F
189 00BF E40D           XRI     0D
190 00C1 9C08           JNZ     ERROR
191 00C3 C601           LD      @1(P2)
192 00C5 C601           LD      @1(P2)
193 00C7 C9FF           ST      -1(P1)
194 00C9 90BF           JMP     $4
195             $SKIP:
196
                        .PAGE   'ERROR PROCESSING'
197                     .LOCAL
198
199             ; PRINT CARRAIGE RETURN , LINE FEED AND LOOP
200             ; TO THE TOP OF THE COMMAND LOOP.
201
202 00CB C401  ERROR:  LDI     H(PUTC)         ; PRINT LINE FEED
```

Figure 2C3–2 (Continued)

```
00CD 37          XPAH    P3
00CE C4C3        LDI     L(PUTC)-1
00D0 33          XPAL    P3
00D1 C40A        LDI     0A
00D3 3F          XPPC    P3
00D4 C43F        LDI     '?'
00D6 3F          XPPC    P3
00D7 C400        LDI     0
00D9 90DB        JMP     LOOP1

                 .PAGE     'HEX NUMBER INPUT'
                 .LOCAL

         ; GHEX GETS A 16-BIT VALUE AND PUSHES IT TO THE STACK.
         ; GHEX2 ASSUMES THE FIRST CHAR IS IN THE E REGISTER.
         ; ONLY THE LAST 4 INPUT DIGITS ARE SAVED.

         ; RETURNS VALUE IN TOP 2 WORDS OF STACK AND TERMINATOR
         ; IN THE AC AND EX REGISTERS.

00DB C401  GHEX2:  LDI     1
00DD 9002          JMP     $6
00DF C400  GHEX:   LDI     0              ; RESET GHEX2 FLAG
00E1 CAFB  $6:     ST      -5(P2)
00E3 C484          LDI     L(GECO)-1      ; SAVE RETURN ADDRESS AND SET
00E5 33            XPAL    P3             ;   TO GECO
00E6 CEFD          ST      @-3(P2)        ; STORE RETURN ADDRESS TO LEAV
00E8 C401          LDI     H(GECO)        ;   FOR RESULT
00EA 37            XPAH    P3
00EB CEFF          ST      @-1(P2)
00ED C2FF          LD      -1(P2)
00EF 9C01          JNZ     $1
00F1 3F            XPPC    P3
00F2 C400  $1:     LDI     0              ; INITIALIZE RESULT TO 0
00F4 CA03          ST      3(P2)
00F6 CA02          ST      2(P2)
00F8 40    $LOOP:  LDE
00F9 03            SCL
00FA FC3A          CAI     '9'+1          ; CHECK FOR 0-9
00FC 940F          JP      $2             ; NOT 0-9, TOO LARGE
00FE 03            SCL
00FF FCF6          CAI     '0'-'9'-1      ; CHECK FOR 0-9
0101 9419          JP      $3             ; IF POSITIVE, NUMBER IS
                                          ;   IN RANGE AND CONVERTED.
0103 C601  $RET:   LD      @1(P2)         ; NUMBER IS NOT A HEX DIGIT,
0105 37            XPAH    P3             ;   RETURN
0106 C601          LD      @1(P2)
0108 33            XPAL    P3
0109 40            LDE
010A 3F            XPPC    P3
010B 90D2          JMP     GHEX
010D 03    $2:     SCL
010E FC0D          CAI     'F'+1-'9'-1    ; CHECK FOR DIGITS A-F.
0110 94F1          JP      $RET           ; NUMBER TOO LARGE
```

Figure 2C3–2 (Continued)

```
256 0112 03            SCL
257 0113 FCFA          CAI     'A'-'F'-1
258 0115 9402          JP      $4              ; DIGIT BETWEEN A&F
259 0117 90EA          JMP     $RET
260 0119 02    $4:     CCL
261 011A F40A          ADI     10              ; ADJUST DIGIT VALUE FOR 10-16
262 011C CAFF  $3:     ST      -1(P2)          ; SAVE ADJUSTED DIGIT
263 011E C404          LDI     4               ; SET UP BIT COUNTER FOR
264 0120 CAFE          ST      -2(P2)          ;   SHIFT.
265 0122 02    $5:     CCL                     ; SHIFT HEX DIGIT LEFT ONE
266 0123 C203          LD      3(P2)           ;   DIGIT, ONE BIT EACH
267 0125 F203          ADD     3(P2)           ;   TIME THROUGH LOOP.
268 0127 CA03          ST      3(P2)
269 0129 C202          LD      2(P2)
270 012B F202          ADD     2(P2)
271 012D CA02          ST      2(P2)
272 012F BAFE          DLD     -2(P2)
273 0131 9CEF          JNZ     $5
274 0133 02            CCL
275 0134 C203          LD      3(P2)           ; ADD CURRENT DIGIT INTO
276 0136 F2FF          ADD     -1(P2)          ;   NUMBER
277 0138 CA03          ST      3(P2)
278 013A 3F            XPPC    P3              ; GET NEXT CHAR
279 013B 90BB          JMP     $LOOP           ; AND LOOP
280
                       .PAGE   'HEX NUMBER OUTPUT'
281                    .LOCAL
282
283            ; PRINT HEX NUMBER WITH TRAILING BLANK (PHEX1) OR
284            ; WITHOUT IT (PHEX2).  NUMBER TO BE PRINTED IS
285            ; IN AC.
286
287 013D CEFF  PHEX1:  ST      @-1(P2)         ; SAVE AC
288 013F C420          LDI     020             ; SET FLAG TO PRINT BLANK AFTE
289 0141 9004          JMP     $1              ;   NUMBER
290 0143 CEFF  PHEX2:  ST      @-1(P2)         ; SAVE AC
291 0145 C400          LDI     0               ; CLEAR FLAG TO PRINT BLANK
292 0147 CEFF  $1:     ST      @-1(P2)         ;   AFTER NUMBER
293 0149 C4C3          LDI     L(PUTC)-1       ; LOAD ADDRESS OF PUTC TO P3
294 014B 33            XPAL    P3              ;   AND SAVE RETURN ADDRESS
295 014C CEFF          ST      @-1(P2)
296 014E C401          LDI     H(PUTC)
297 0150 37            XPAH    P3
298 0151 CEFF          ST      @-1(P2)
299 0153 C402          LDI     2               ; SET FLAG FOR 1ST NUMBER
300 0155 CEFF          ST      @-1(P2)
301 0157 C204          LD      4(P2)           ; GET ORIGINAL VALUE
302 0159 1C            SR                      ; SHIFT TO LOW 4 BITS
303 015A 1C            SR
304 015B 1C            SR
305 015C 1C            SR
306 015D 02    $5:     CCL                     ; CONVERT TO ASCII
307 015E F4F6          ADI     -10
308 0160 9404          JP      $2              ; NUMBER IS A THRU F
```

Figure 2C3–2 (Continued)

```
309 0162 F43A           ADI     '0'+10
310 0164 9002           JMP     $3
311 0166 F440  $2:      ADI     'A'-1           ; THE -1 TAKES CARE OF CARRY I
312 0168 3F    $3:      XPPC    P3              ; PRINT NUMBER
313 0169 BA00           DLD     (P2)
314 016B 9806           JZ      $4
315 016D C204           LD      4(P2)           ; GET ORIGINAL NUMBER
316 016F D40F           ANI     0F              ; MASK 2ND DIGIT
317 0171 90EA           JMP     $5
318 0173 C203  $4:      LD      3(P2)           ; CHECK FOR PRINTING BLANK
319 0175 9801           JZ      $6
320 0177 3F             XPPC    P3              ; IF NOT 0, PRINT BLANK
321 0178 C201  $6:      LD      1(P2)           ; RESTORE RETURN ADDRESS
322 017A 37             XPAH    P3
323 017B C202           LD      2(P2)
324 017D 33             XPAL    P3
325 017E C604           LD      @4(P2)          ; RESTORE STACK AND AC
326 0180 C601           LD      @1(P2)
327 0182 3F             XPPC    P3              ; RETURN
328 0183 90B8           JMP     PHEX1
329
                        .PAGE   'GECO'
330                     .LOCAL
331
332             ; GECO IS USED FOR KEYBOARD INPUT SO IT ECHOS THE
333             ; CHARACTER BUT DOES NOT ENABLE THE READER RELAY.
334
335 0185 C408  GECO:    LDI     8               ; SET COUNT = 8
336 0187 CAFF           ST      -1(P2)
337 0189 06    $2:      CSA                     ; WAIT FOR START BIT
338 018A D420           ANI     020
339 018C 9CFB           JNZ     $2              ; NOT FOUND
340 018E C457           LDI     87              ; DELAY 1/2 BIT TIME
341 0190 8F04           DLY     4
342 0192 06             CSA                     ; IS START BIT STILL THERE?
343 0193 D420           ANI     020
344 0195 9CF2           JNZ     $2              ; NO
345 0197 06             CSA                     ; SEND START BIT (NOTE THAT
346 0198 DC01           ORI     1               ;   OUTPUT IS INVERTED)
347 019A 07             CAS
348 019B C47E  $LOOP:   LDI     126             ; DELAY 1 BIT TIME
349 019D 8F08           DLY     8
350 019F 06             CSA                     ; GET BIT (SENSEB)
351 01A0 D420           ANI     020
352 01A2 9802           JZ      $3
353 01A4 C401           LDI     1
354 01A6 CAFE  $3:      ST      -2(P2)          ; SAVE BIT VALUE (0 OR 1)
355 01A8 1F             RRL                     ; ROTATE INTO LINK
356 01A9 01             XAE
357 01AA 1D             SRL                     ; SHIFT INTO CHARACTER
358 01AB 01             XAE                     ; RETURN CHAR TO E
359 01AC 06             CSA                     ; ECHO BIT TO OUTPUT
360 01AD DC01           ORI     1
361 01AF E2FE           XOR     -2(P2)
```

Figure 2C3–2 (Continued)

```
362 01B1 07             CAS
363 01B2 BAFF           DLD     -1(P2)          ; DECREMENT BIT COUNT
364 01B4 9CE5           JNZ     $LOOP           ; LOOP UNTIL 0
365 01B6 06             CSA                     ; SET STOP BIT
366 01B7 D4FE           ANI     0FE
367 01B9 07             CAS
368 01BA 8F08           DLY     8
369 01BC 40             LDE                     ; AC HAS INPUT CHARACTER
370 01BD D47F           ANI     07F
371 01BF 01             XAE
372 01C0 40             LDE
373 01C1 3F             XPPC    P3              ; RETURN
374 01C2 90C1           JMP     GECO
375
                        .PAGE   'PUTC'
376                     .LOCAL
377
378             ; PUT CHARACTER  IN AC TO TTY.   ALL REGS SAVED.
379             ; IF INPUT DETECTED, CONTROL PASSES TO PROMPT.
380             ; NOTE: TTY LOGIC LEVELS ARE INVERTED FOR OUTPUT
381
382 01C4 01     PUTC:   XAE
383 01C5 C4FF           LDI     255
384 01C7 8F17           DLY     23
385 01C9 06             CSA                     ; SET OUTPUT BIT TO LOGIC 0 FL
386 01CA DC01           ORI     1               ;   FOR START BIT.  (NOTE INVER
387 01CC 07             CAS
388 01CD C409           LDI     9               ; INITIALIZE BIT COUNT
389 01CF CAFF           ST      -1(P2)          ;AT 0FF5
390 01D1 C48A   $1:     LDI     138             ; DELAY 1 BIT TIME
391 01D3 8F08           DLY     8
392 01D5 BAFF           DLD     -1(P2)          ; DECREMENT BIT COUNT.
393 01D7 9810           JZ      $EXIT
394 01D9 40             LDE                     ; PREPARE NEXT BIT
395 01DA D401           ANI     1
396 01DC CAFE           ST      -2(P2)          ;AT 0FF4
397 01DE 01             XAE                     ; SHIF DATA RIGHT 1 BIT
398 01DF 1C             SR
399 01E0 01             XAE
400 01E1 06             CSA                     ; SET UP OUTPUT BIT
401 01E2 DC01           ORI     1
402 01E4 E2FE           XOR     -2(P2)
403 01E6 07             CAS                     ; PUT BIT TO TTY
404 01E7 90E8           JMP     $1
405 01E9 06     $EXIT:  CSA                     ; SET STOP BIT
406 01EA D4FE           ANI     0FE
407 01EC 07             CAS
408 01ED D420           ANI     020             ; CHECK FOR KEYBOARD INPUT SEN
409 01EF 9803           JZ      $2              ; ATTEMPTED INPUT (NOTE THAT
410                                             ;   INPUT IS NOT INVERTED)
411 01F1 3F             XPPC    P3              ; RETURN
412 01F2 90D0           JMP     PUTC
413 01F4 C400   $2:     JS      P3,CMDLP
```

Figure 2C3–2 (Continued)

```
    01F6 37C4
    01F8 4133
    01FA 3F
414       0000          . END

            AC      FFFE          CMDLP   0042          ENTER   0028
            ERROR   00CB          EX      FFFF          EXIT    000B
            EXOFF   FFFF          GECO    0185          GHEX    00DF
            GHEX2   00DB          GO      005B *        LOOP1   00B6
            MOD     006C *        P1      0001          P2      0002
            P2ADR   0FF6          P3      0003          PC      FFF8
            PHEX1   013D          PHEX2   0143          PT1     FFFA
            PT2     FFFC          PUTC    01C4          SR      0000
            STACK   0FFF          START   0001 *        TYPE    0067 *
            $1      0077          $1      00F2          $1      0147
            $1      01D1          $2      0075          $2      010D
            $2      0166          $2      0189          $2      01F4
            $3      011C          $3      0168          $3      01A6
            $4      008A          $4      0119          $4      0173
            $5      0122          $5      015D          $6      00E1
            $6      0178          $EXIT   01E9          $LOOP   00F8
            $LOOP   019B          $RET    0103          $SKIP   0067
            $SKIP   00CB


            NO ERROR LINES
            SOURCE CHECKSUM=E86E
            FIRST INPUT SECTOR HEX -  0290
            FINAL INPUT SECTOR HEX -  02A0
```

NS10561

Figure 2C3–2 (Concluded)

```
 1                          .TITLE  DISP, 'MOVING MESSAGE FOR MULTI '
 2
 3                  ;       MESSAGE MUST BE > 32 CHARACTERS.
 4
 5
 6                  ;       3/17/76
 7
 8
 9                  ;       RAM LOCATIONS USED.
10
11
12                  ;       0F00    ADDRESS OF MESSAGE HIGH
13                  ;       0F01    ADDRESS OF MESSAGE LOW
14                  ;       0F02    NEXT ADDRESS OF MESSAGE HIGH
15                  ;       0F03    NEXT ADDRESS OF MESSAGE LOW
16                  ;       0F04    CHAR PER LINE COUNTER
17
18
19
20          0000    HST     =       0               ;SAME AS 0F00
21          0001    LST     =       1               ;SAME AS 0F01
22          0002    HTMP    =       2               ;SAME AS 0F02
23          0003    LTMP    =       3               ;SAME AS 0F03
24          0004    COUNT   =       4               ;SAME AS 0F04
25          0800    ADR     =       0800            ;ADDRESS OF DISPLAY.
26          0F00    RAM     =       0F00            ;START OF RAM.
27
28          0400            =       0400            ;STARTING ADDRESS.
29
30
31                  ;       MESSAGE IS ASCII STRING IN MEMORY.
32                  ;       END OF MESSAGE IS A BYTE OF ALL 0.
33
34
                            .PAGE
35
36                  START:
37  0400 08                 NOP
38  0401 C420               LDI     020             ;SET F1, TURN ON SC/MP 1.
39  0403 07                 CAS                     ;SEND FLAG.
40  0404 C40F               LDI     H(RAM)          ;PUT RAM ADDRESS IN P3.
41  0406 37                 XPAH    3
42  0407 C400               LDI     L(RAM)
43  0409 33                 XPAL    3
44  040A C404               LDI     H(MMSG)         ;SET STARTING ADDRESS IF MESSA
45  040C CB00               ST      HST(3)          ;SAVE IN RAM.
46  040E C477               LDI     L(MMSG)
47  0410 CB01               ST      LST(3)
48  0412 C408               LDI     H(ADR)          ;PUT ADDRESS OF DISPLAY IN P1.
49  0414 35                 XPAH    1
50  0415 C400               LDI     L(ADR)
51  0417 31                 XPAL    1
52  0418 C400               LDI     0               ;CLEAR DISPLAY.
53  041A C900               ST      (1)
54  041C C420               LDI     32              ;SET LINE COUNT.
55  041E CB04               ST      COUNT(3)
56                  S:
57  0420 C301               LD      LST(3)          ;PUT ADDRESS IN TEMP.
58  0422 CB03               ST      LTMP(3)
```

Figure 2C3–3. Program Listing for Burroughs Self-Scan Routine

```
 59 0424 32          XPAL    2               ;SET P2 TO ADDRESS.
 60 0425 C300        LD      HST(3)
 61 0427 CB02        ST      HTMP(3)
 62            PRNT:
 63 0429 36          XPAH    2               ;HIGH ADDRESS IN P2.
 64            PRINT:
 65 042A C601        LD      @1(2)           ;GET NEXT WORD.
 66 042C 9C22        JNZ     SELF            ;CHECK IF DONE.
 67 042E C302        LD      HTMP(3)         ;RESTORE POINTER.
 68 0430 36          XPAH    2
 69 0431 C303        LD      LTMP(3)
 70 0433 32          XPAL    2
 71 0434 900D        JMP     MORE
 72            ON:
 73 0436 32          XPAL    2               ;SAVE IN P2 LOW.
 74 0437 C302        LD      HTMP(3)         ;RESTORE HIGH.
 75 0439 90EE        JMP     PRNT
 76            NEW:
 77 043B C420        LDI     32              ;SAVE LINE COUNT.
 78 043D CB04        ST      COUNT(3)
 79 043F C4FF        LDI     0FF             ;DO A SHORT DELAY.
 80 0441 8F80        DLY     080
 81            MORE:
 82 0443 C200        LD      (2)             ;CHECK IF DONE.
 83 0445 98D9        JZ      S
 84 0447 AB03        ILD     LTMP(3)         ;BUMP RAM POINTER.
 85 0449 9CEB        JNZ     ON
 86 044B 32          XPAL    2               ;NEXT ADDRESS.
 87 044C AB02        ILD     HTMP(3)         ;BUMP HIGH.
 88 044E 90DA        JMP     PRINT
 89            SELF:
 90 0450 01          XAE                     ;SAVE CHAR.
 91 0451 40          LDE                     ;GET CHAR.
 92 0452 02          CCL                     ;CLEAR LINK.
 93 0453 F4E0        ADI     0E0             ;CHECK IF LESS THAN 020.
 94 0455 9402        JP      GT1F            ;NO > 01F.
 95 0457 90D1        JMP     PRINT           ;LESS THAN 01F RETURN.
 96            GT1F:
 97 0459 02          CCL                     ;CLEAR LINK.
 98 045A F4C0        ADI     0C0             ;CHECK IF > 05F.
 99 045C 94CC        JP      PRINT           ;YES RETURN.
100 045E 40          LDE                     ;CHAR IS VALID.
101 045F D43F        ANI     03F             ;STRIP OFF HIGH BITS.
102 0461 DC80        ORI     080             ;SET CLEAR AND DISPLAY BITS.
103 0463 C900        ST      (1)             ;SEND WORD.
104 0465 06          CSA
105 0466 DC01        ORI     1               ;SET WRITE CYCLE FLAG 0.
106 0468 07          CAS                     ;NOW SET FLAG 0.
107 0469 D4FE        ANI     0FE             ;NOW RESET FLAG.
108 046B 07          CAS                     ;DO IT.
109           DOWAIT:
110 046C 06          CSA                     ;GET STATUS.
111 046D D420        ANI     020             ;CHECK IF SENSE B IS SET.
112 046F 9CFB        JNZ     DOWAIT          ;WAIT IF SET.
113 0471 BB04        DLD     COUNT(3)        ;BUMP COUNTER.
114 0473 98C6        JZ      NEW
115 0475 90B3        JMP     PRINT
116
117           MMSG:
```

Figure 2C3–3 (Continued)

```
118 0477 2020           .ASCII              MULTI PROCESSOR OPERATION OF '
    0479 2020
    047B 2020
    047D 2020
    047F 2020
    0481 4D55
    0483 4C54
    0485 4920
    0487 5052
    0489 4F43
    048B 4553
    048D 534F
    048F 5220
    0491 4F50
    0493 4552
    0495 4154
    0497 494F
    0499 4E20
    049B 4F46
    049D 20
119 049E 5457           .ASCII  'TWO SC/MPS, ONE RUNNING KITBUG THE OTHER'
    04A0 4F20
    04A2 5343
    04A4 2F4D
    04A6 5053
    04A8 2C20
    04AA 4F4E
    04AC 4520
    04AE 5255
    04B0 4E4E
    04B2 494E
    04B4 4720
    04B6 4B49
    04B8 5442
    04BA 5547
    04BC 2054
    04BE 4845
    04C0 204F
    04C2 5448
    04C4 4552
120 04C6 2052           .ASCII  ' RUNNING A BURROUGHS SELF SCAN DISPLAY.
    04C8 554E
    04CA 4E49
    04CC 4E47
    04CE 2041
    04D0 2042
    04D2 5552
    04D4 524F
    04D6 5547
    04D8 4853
    04DA 2053
    04DC 454C
    04DE 4620
    04E0 5343
    04E2 414E
    04E4 2044
    04E6 4953
    04E8 504C
    04EA 4159
```

Figure 2C3–3 (Continued)

```
      04EC 2E20
      04EE 2020
      04F0 20
  121 04F1 00              BYTE   0
  122
  123
  124      0000            END


DISP  MOVING MESSAGE FOR MULTI


ADR      0800          COUNT    0004          DOWAIT   046C
GT1F     0459          HST      0000          HTMP     0002
LST      0001          LTMP     0003          MMSG     0477
MORE     0443          NEW      043B          ON       0436
PRINT    042A          PRNT     0429          RAM      0F00
S        0420          SELF     0450          START    0400 *


NO ERROR LINES
SOURCE CHECKSUM=D136
```

NS10562

Figure 2C3-3 (Concluded)

INTERFACING A SC/MP SYSTEM WITH A CASSETTE RECORDER

General Description

Figure 2C4-1 shows how a casette tape recorder can be interfaced with a SC/MP-based system to provide approximately 40K 8-bit bytes of data on each side of a 30-minute tape. As an alternative to using paper tape, PROMs, or more complex and expensive media, the cassette interface can be used to store and transport program libraries; also, systems of similar design could be used for small-business inventories and many other applications.

Off-the-shelf integrated circuits were used to implement the system shown in figure 2C4-1. A recorder in the cost range of 50 dollars will provide satisfactory performance; however, a higher priced ($80 to $100) recorder should provide better consistency and greater reliability. The recorders listed in table 2C4-1 were used to verify the accuracy and reliability of this application. Using the four recorders, a 10K-byte program was loaded 10 times in succession and then two playbacks were made from each recorder. A different recorder was used to record and playback and, in all cases, the results were error-free. The transmission rate of 330 baud (40 bytes/second) is sufficient for most applications; the clocking scheme permits the storage of 17 2K-byte programs (including interprogram gaps) to be recorded on each 15-minute side of the cassette tape.

Table 2C4-1. Cassette Recorders Used for Accuracy and Reliability Tests

Make	Model	Approximate Cost
Panasonic	RQ309AS	$ 40
Panasonic	RQ423S	$ 70
Sony	TC-40	$100
Sony	TC-55	$155

*Specifying the ideal recorder for a particular application is difficult; it is recommended that the output waveform (WF-A, figure 2C4-3) be used as a guide to selection.

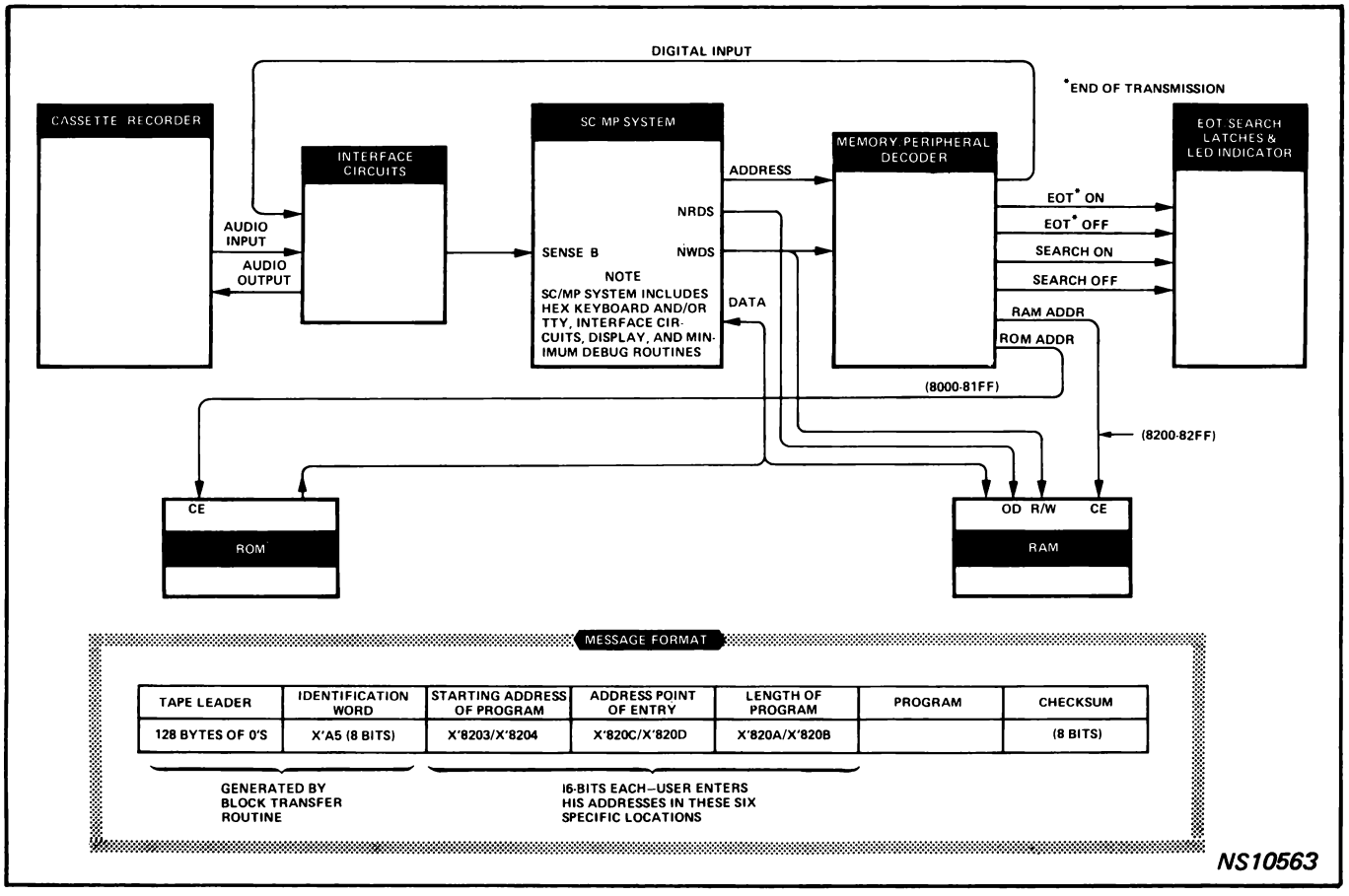


Figure 2C4–1. Cassette Recorder Interfaced with SC/MP System–Block Diagram and Message Format

Operator Control

An input/output program residing in PROM provides all timing and control functions required to send and receive information between the CPU and the recorder. The send operation can be summarized as follows. Using a keyboard or a tape reader, the operator loads RAM locations X'8203 (high-order byte) and X'8204 (low-order byte) with a 4-digit hexadecimal address that corresponds to the starting address of the program – see MESSAGE FORMAT in figure 2C4-1. Next, RAM locations X'820C (high-order byte) and X'820D (low-order byte) are loaded with the POINT OF ENTRY ADDRESS. This entry point may be the starting address of any program to which the operator wants to transfer control upon completion of loading (playback mode). Finally, the high and low order bytes for LENGTH OF PROGRAM are loaded into RAM locations X'820A and X'820B, respectively. To initiate the output cycle, the operator turns on the recorder and transfers control to address X'80C7 – the beginning of the data write routines. After the TAPE LEADER (128 bytes of zeros) is run through, the Search (LED) Indicator lights and stays on until the data transmission is complete; at this time, the Search Indicator is turned off, the End-of-Transmission Indicator is turned on, and the program halts at location X'8142. *NOTE: If the recorder is inadvertently stopped before the output cycle is completed, the tape must be rewound and the cycle restarted.*

In the receive (playback) mode, the operator transfers control via the keyboard to X'8000 – the starting address of the bootstrap loader routine. The bootstrap loader conditions SC/MP and then addresses the receive routine. Now, when the cassette is turned on and the operator selects the playback mode, the Search Indicator is turned on until the IDENTIFICATION WORD (figure 2C4-1) is recognized; then, the indicator is turned off. *NOTE: For normal operation, the Search Indicator should be "on" from 3 to 5 seconds; if it is on for a longer period because of defective tape, dirty heads, or another malfunction, the operator should abort the operation and restart it at X'8000.* If the checksum is good, the Search Indicator is turned on again when the transmission-of-data is completed.

When in the playback mode, the volume control of recorder should be adjusted until the "monitor output" is just below the clipping level when measured with an oscilloscope. Using the Search Indicator on-off time of approximately 5 seconds as a limit switch, the adjustment can be optimized by a trial-and-error method.

System Operation

Transmission of data from SC/MP to the cassette recorder is accomplished by using a scheme that is self-synchronizing on a bit-time basis; that is, data are referenced to a "time-frame" rather than a leading or trailing pulse edge; thus, there is no cumulative error buildup in the system. The "approximate" 4-millisecond time frame (duration between clock pulses) is established by the send routine. A logic '0' is represented by the absence of a pulse at the midpoint of the time frame; a logic '1' is represented by the presence of a pulse. The clock and bit (data) pulses are generated by the address decoder circuits shown in figure 2C4-2. To generate either a clock or a bit (logic '1') pulse, a unique address is presented to the system address bus during the execution of a STore Instruction by SC/MP. The clock or bit pulse is then transmitted to the cassette recorder via the interface circuits shown in figure 2C4-3. A negative-going pulse is produced to begin the time frame. If the data bit is a '1', the decoder is addressed at the midpoint (between clock pulses) of the time frame and a second negative-going pulse is generated. If the data bit is a '0', the decoder is not addressed and no pulse appears between the clock pulses of the time-frame.

To record, the data write routines generate a long leader of zeros plus the identification word (X'A5) shown in figure 2C4-1; these data are presented to the interface circuits in figure 2C4-3. The leader allows the tape-drive motor and AGC loop to stabilize; the leader also serves as an interprogram gap that facilitates multiple-program recording on a single side of the cassette tape. As shown by the idealized waveforms in figure 2C4-3, TTL inputs from the address decoder are changed to signals that are suitable for recording on the tape. User data are transmitted following the identification word.

In the playback mode of operation, the recorder output (WF-A, figure 2C4-3) is translated to a TTL signal (WF-B) that drives the Sense B input (WF-C) of SC/MP. The processor tests the Sense B line (output of "send" latch) for a logic '0', which occurs at the first clock pulse of the time frame. The latch is reset by SC/MP and, after a predetermined delay (approximately one-half time frame), Sense B is again tested. If a negative-going pulse is found at the "center" of the time frame, the data bit is recognized as a logic '1' and the latch is reset. If there is no zero-transition between the first and last pulses of the time frame, the bit is recognized as a logic '0'. As previously indicated, this technique – referencing the data bit to a time interval rather than to the leading or trailing edge of a pulse – prevents cumulative error buildup in the system. Tape format is such that upon completion of loading a program from the cassette, the program may be executed or control can be transferred to another existing program; for instance, a debug program. As previously indicated, the user can transfer program control by simply loading the POINT OF ENTRY ADDRESS (figure 2C4-1) with the proper starting location.

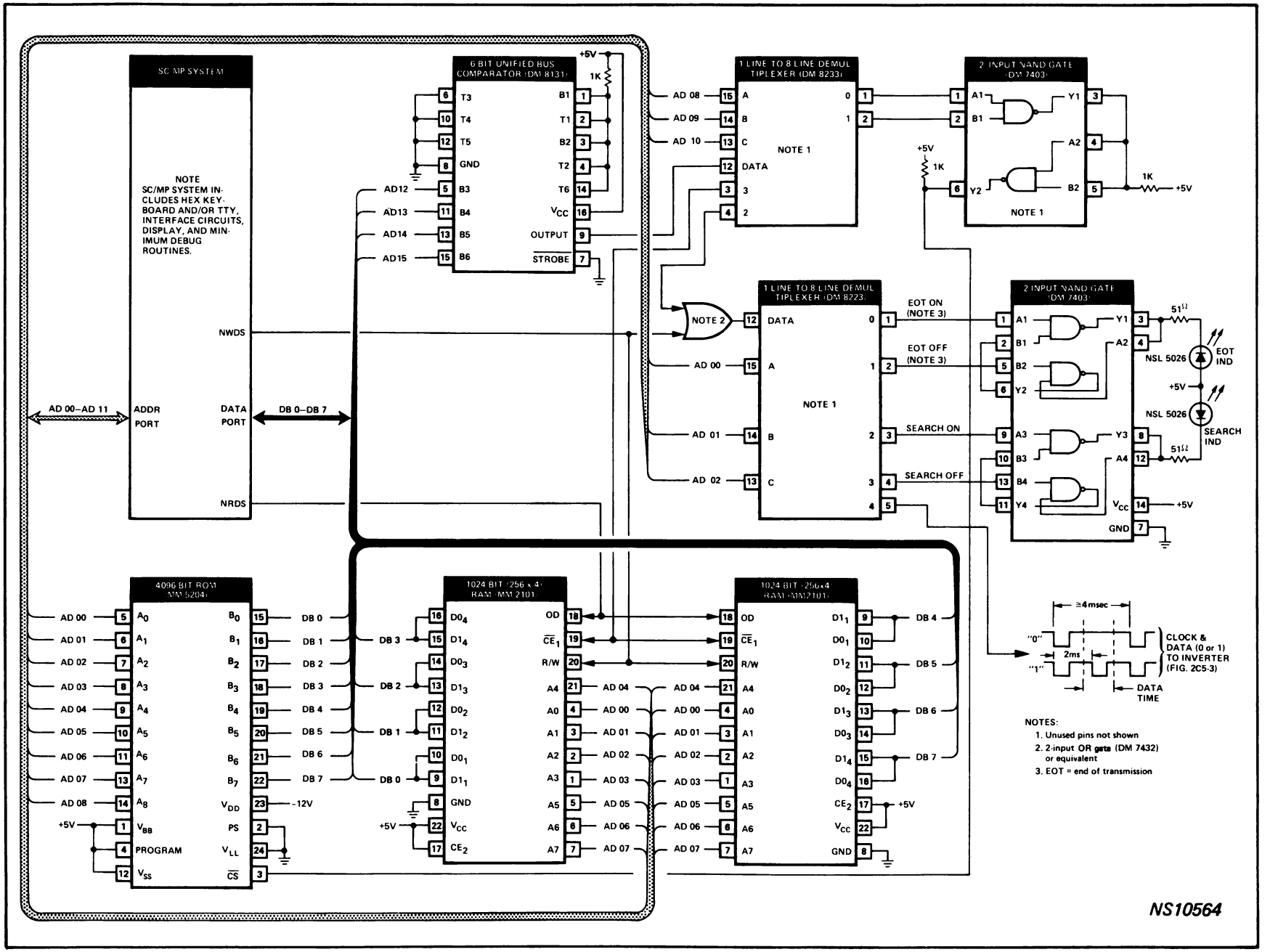


Figure 2C4–2. Decoding and Memory-Interface Circuits

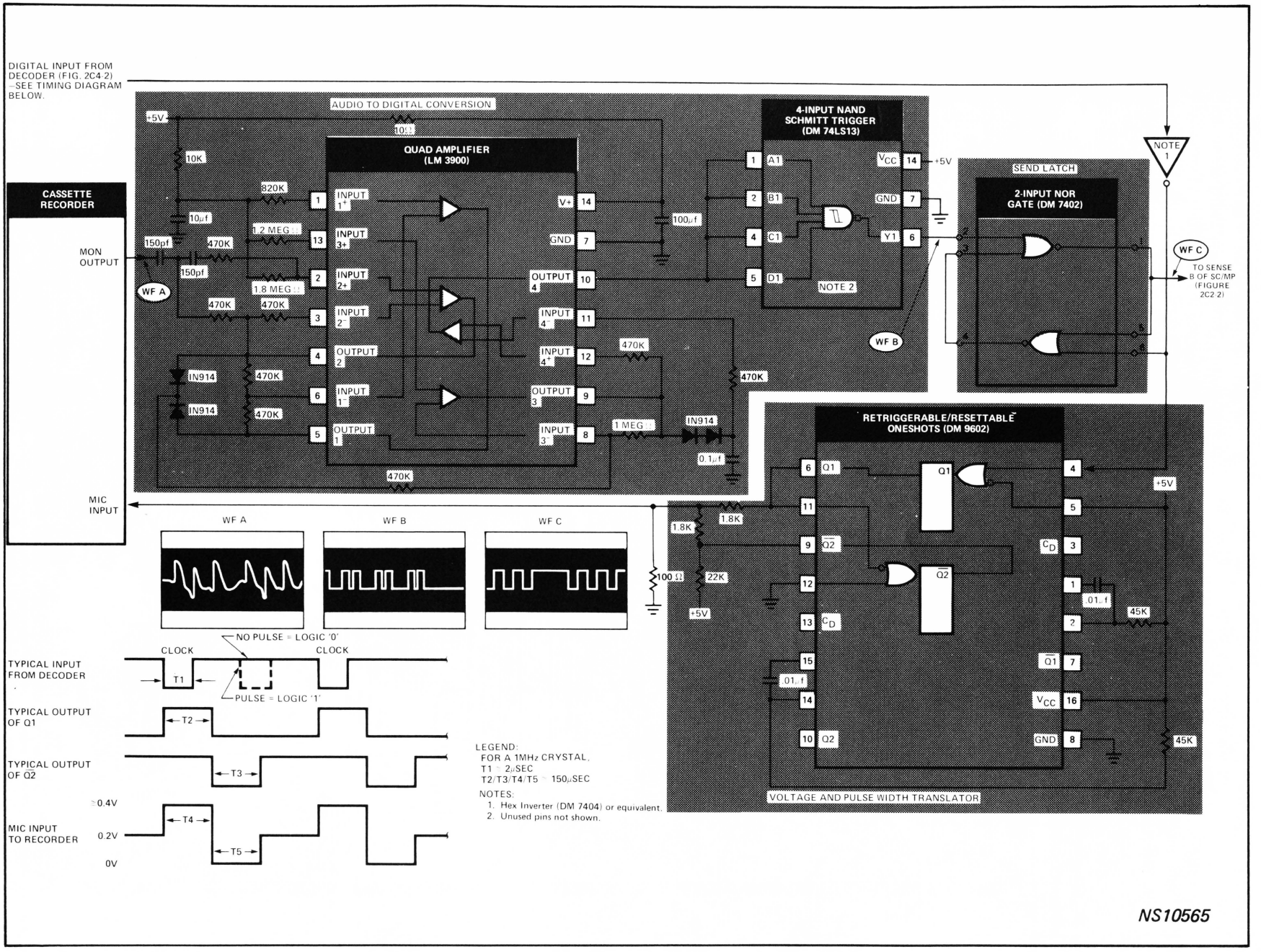


Figure 2C4–3. Cassette-to-SC/MP Interface Circuits

Software Considerations

The "data write" and "data receive" routines are stored in ROM. A "minimum control" routine is also stored in ROM; this routine directs communications between the operator and a "hex" keyboard, and controls the LED indicators shown in figure 2C4-2. Send and receive flowcharts and a complete program listing for the cassette-to-SC/MP interface are shown in figures 2C4-4 and 2C4-5, respectively.

To transfer control to RAM or to modify locations in RAM, an input/output interface capability is required by the operator. Such a capability is provided by the SC/MP Low Cost Development System or by the SC/MP minimum debug kit with a "hex" keyboard/display and the necessary interface circuits. *NOTE: Refer to figures 2C2-8 and 2C2-14 for guidelines to design a keyboard-entry/display interface.)*

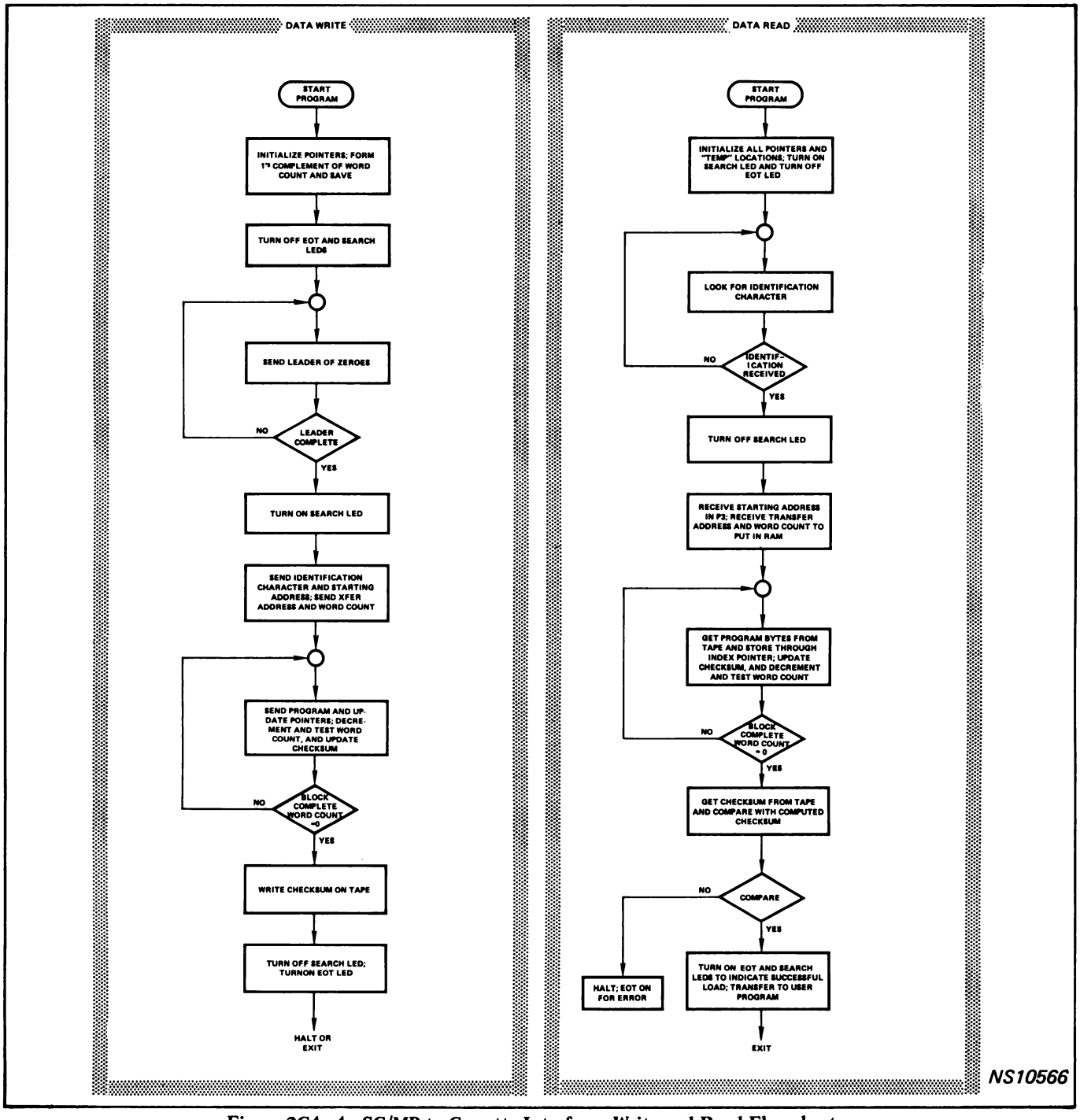


Figure 2C4–4. SC/MP-to-Cassette Interface–Write and Read Flowcharts

```
 1                      TITLE  TAPEIO, ' SC/MP ROUTINES'
 2
 3       8000           .=X'8000
 4
 5       8200  RAM      =        X'8200    ; RAM ADDRESS FOR POINTER
 6       8300  PERIPH   =        X'8300    ; PERIPHERAL ADDRESS FOR POINT
 7
 8       0003  P3       =        3         ; POINTER #3
 9       0002  P2       =        2         ; POINTER #2
10       0001  P1       =        1         ; POINTER #1
11
12             ; TEMPORARY DATA IN RAM
13
14       0000  CNTU     =        0         ; INSIDE COUNTER FOR LEADER
15       0001  CNTL     =        1         ; OUTSIDE COUNTER FOR LEADER
16       0002  CKSUM    =        2         ; CHECK SUM COUNTER
17       0003  STARTU   =        3         ; STARTING ADDRESS (UPPER)
18       0004  STARTL   =        4         ; STARTING ADDRESS (LOWER)
19       0005  BITCNT   =        5         ; BIT COUNTER
20       0006  TEMP1    =        6         ; TEMPORARY STORAGE LOCATIONS
21       0007  TEMP2    =        7         ;      "         "         "
22       0008  TEMP3    =        8         ;      "         "         "
23       0009  TEMP4    =        9         ;      "         "         "
24       000A  WDCNTU   =        10        ; WORD COUNT (UPPER)
25       000B  WDCNTL   =        11        ; WORD COUNT (LOWER)
26       000C  JUMPU    =        12        ; TRANSFER ADDRESS (UPPER)
27       000D  JUMPL    =        13        ; TRANSFER ADDRESS (LOWER)
28
29             ; PERIPHERAL ORDER CODES
30
31       0000  EOTON    =        0         ; END OF TAPE LED ON POINTER
32       0001  EOTOFF   =        1         ; END OF TAPE LED OFF POINTER
33       0002  SRCHON   =        2         ; SEARCH LED ON POINTER
34       0003  SRCHOF   =        3         ; SEARCH LED OFF POINTER
35       0004  FLAG     =        4         ; READ/WRITE FLAG
36
37
```

Figure 2C4–5. SC/MP-to-Cassette Interface–Program Listing

```
                        .PAGE 'BOOTSTRAP LOADER'
38
39
40              ; BOOTSTRAP LOADER ROUTINE.  RECEIVES PROGRAM FROM TAPE.
41              ; ALL NECCESSARY INFORMATION FOR LOADING IS ON TAPE.
42
43              ; THIS PROGRAM MAY BE REASSEMBLED TO ADDRESS X'0000 TO
44              ; FUNCTION AS A POWER-ON LOADER.
45
46              ; INPUT OPERATION OF LED INDICATORS;
47
48              ;       SEARCH LED ON WHEN PROGRAM STARTS
49                      SEARCH LED OFF WHEN IDENTIFIER CHARACTER RECEIVED
50                      END OF TAPE (EOT) LED ON WHEN RECEPTION COMPLETE
51                      SEARCH LED ON IF CHECKSUMS COMPARE
52
53              ;       CONTROL IS THEN TRANSFERED TO USER PROGRAM
54
55
56 8000 08    BOOT:   NOP                     ; FOR RELOCATION TO X'0000
57 8001 C400           LDI     L(RAM)          ; INITIALIZE RAM POINTER
58 8003 32             XPAL    P2              ; IN P2
59 8004 C482           LDI     H(RAM)
60 8006 36             XPAH    P2
61 8007 C400           LDI     0               ; CLEAR ACCUMULATOR
62 8009 CA02           ST      CKSUM(P2)       ; INITIALIZE CHECKSUM COUNTER
63 800B C400           LDI     L(PERIPH)       ; PUT PERIPHERAL POINTER
64 800D 33             XPAL    P3              ; IN P3
65 800E C483           LDI     H(PERIPH)
66 8010 37             XPAH    P3
67 8011 CB02           ST      SRCHON(P3)      ; TURN ON SEARCH LED
68 8013 CB01           ST      EOTOFF(P3)      ; TURN OFF END OF TAPE LED
69 8015 C400           LDI     0               ; CLEAR ACCUMULATOR
70 8017 01             XAE                     ; CLEAR E REGISTER
71 8018 C48F           LDI     L(GETBIT)-1     ; PLACE ADDRESS OF GET BIT
72 801A 31             XPAL    P1              ; IN P1
73 801B C480           LDI     H(GETBIT)
74 801D 35             XPAH    P1
75 801E 3D    LOCID:   XPPC    P1              ; GO TO GETBIT FOR INPUT
76 801F 40             LDE
77 8020 E4A5           XRI     X'A5            ; CHECK FOR PROPER ID CHARACTE
78 8022 9802           JZ      SETPNT          ; IF ID RECEIVED, TAKE REST OF
79 8024 90F8           JMP     LOCID           ; PROGRAM, ELSE GET NEXT BIT
80 8026 CB03  SETPNT:  ST      SRCHOF(P3)      ; TURN OFF SEARCH LED
81 8028 C46D           LDI     L(RECV)-1       ; PLACE ADDRESS OF BYTE RECEIV
82 802A 31             XPAL    P1              ; IN P1
83 802B C480           LDI     H(RECV)
84 802D 35             XPAH    P1
85 802E 3D             XPPC    P1              ; GET STARTING ADDRESS (LOWER)
86 802F 33             XPAL    P3              ; AND PLACE IN P3
87 8030 3D             XPPC    P1              ; GET STARTING ADDRESS (UPPER)
88 8031 37             XPAH    P3
89 8032 3D             XPPC    P1              ; GET TRANSFER ADDRESS AND
90 8033 CA0D           ST      JUMPL(P2)       ; SAVE IN RAM
91 8035 3D             XPPC    P1
```

Figure 2C4–5 (Continued)

```
 92 8036 CA0C           ST      JUMPU(P2)
 93 8038 3D             XPPC    P1              ; GET WORD COUNT (LOWER)
 94 8039 CA0B           ST      WDCNTL(P2)
 95 803B 3D             XPPC    P1              ; GET WORD COUNT (UPPER)
 96 803C CA0A           ST      WDCNTU(P2)
 97
 98 803E 3D     BOOTIN: XPPC    P1              ; GO TO RECEIVE
 99 803F CF01           ST      @1(P3)          ; STORE AND INCREMENT POINTER
100 8041 F202           ADD     CKSUM(P2)       ; ADD CHARACTER TO CHECKSUM
101 8043 CA02           ST      CKSUM(P2)
102 8045 AA0B           ILD     WDCNTL(P2)      ; INCREMENT LOWER WORD COUNTER
103 8047 9CF5           JNZ     BOOTIN          ; CHECK FOR ZERO
104 8049 AA0A           ILD     WDCNTU(P2)      ; INCREMENT UPPER WORD COUNTER
105 804B 9CF1           JNZ     BOOTIN          ; CHECK FOR END OF TRANSMISSIO
106 804D 3D             XPPC    P1              ; GET CHECKSUM FROM TAPE
107 804E E202           XOR     CKSUM(P2)       ; COMPARE TO CALCULATED VALUE
108 8050 9809           JZ      EXECPR          ; EXECUTE LOADED PROGRAM
109 8052 C400           LDI     L(PERIPH)
110 8054 33             XPAL    P3
111 8055 C483           LDI     H(PERIPH)
112 8057 37             XPAH    P3
113 8058 CB00           ST      EOTON(P3)       ; TURN ON EOT LED TO INDICATE
114 805A 00             HALT                    ; CHECKSUM ERROR AND HALT
115
116 805B C400   EXECPR: LDI     L(PERIPH)
117 805D 33             XPAL    P3
118 805E C483           LDI     H(PERIPH)
119 8060 37             XPAH    P3
120 8061 CB00           ST      EOTON(P3)       ; TURN ON END OF TAPE LED
121 8063 CB02           ST      SRCHON(P3)      ; TURN ON SEARCH LED
122 8065 C20D           LD      JUMPL(P2)       ; LOAD TRANSFER ADDRESS
123 8067 33             XPAL    P3
124 8068 C20C           LD      JUMPU(P2)
125 806A 37             XPAH    P3
126 806B C7FF           LD      @-1(P3)         ; DECREMENT POINTER FOR FETCH
127 806D 3F             XPPC    P3              ; EXECUTE
128
129
130             RECEIVE ROUTINE.  RECEIVES ONE 8-BIT CHARACTER INTO
131             ACCUMULATOR.
132
133
134 806E C48F   RECV:   LDI     L(GETBIT)-1     ; PLACE ADDRESS OF GETBIT
135 8070 31             XPAL    P1              ; IN P1
136 8071 CA07           ST      TEMP2(P2)       ; SAVE CURRENT CONTENTS OF P1
137 8073 C480           LDI     H(GETBIT)
138 8075 35             XPAH    P1
139 8076 CA06           ST      TEMP1(P2)
140 8078 C408           LDI     8               ; SET BIT COUNT
141 807A CA05           ST      BITCNT(P2)
142 807C C400           LDI     0               ; CLEAR ACCUMULATOR
143 807E 01             XAE                     ; CLEAR E REGISTER
144 807F 3D     LOOP:   XPPC    P1              ; GO TO GETBIT
145 8080 BA05           DLD     BITCNT(P2)      ; DECREMENT BIT COUNT
146 8082 9802           JZ      RETRN2          ; CHECK FOR ZERO
```

Figure 2C4–5 (Continued)

```
147 8084 90F9           JMP     LOOP
148 8086 C207  RETRN2:  LD      TEMP2(P2)       ; RESTORE P1 TO ORIGINAL
149 8088 31             XPAL    P1              ; CONTENTS
150 8089 C206           LD      TEMP1(P2)
151 808B 35             XPAH    P1
152 808C 40             LDE                     ; PLACE CHARACTER IN ACC.
153 808D 3D             XPPC    P1              ; RETURN
154 808E 90DE           JMP     RECV
155
156
157             ; GET BIT ROUTINE.  RECEIVES 1 BIT INTO E REGISTER
158
159
160 8090 C400  GETBIT:  LDI     L(PERIPH)       ; PLACE PERIPHERAL ADDR.  IN P3
161 8092 33             XPAL    P3
162 8093 CA09           ST      TEMP4(P2)       ; SAVE ORIGINAL CONTENTS OF P3
163 8095 C483           LDI     H(PERIPH)
164 8097 37             XPAH    P3
165 8098 CA08           ST      TEMP3(P2)
166 809A 19             SIO                     ; SHIFT E REGISTER
167 809B 06     CKSA:   CSA                     ; COPY STATUS TO ACCUMULATOR
168 809C D420           ANI     X'20            ; MASK
169 809E 9802           JZ      CLOCK           ; IF ZERO, BIT RECEIVED
170 80A0 90F9           JMP     CKSA            ; CHECK AGAIN
171 80A2 C400  CLOCK:   LDI     0               ; CLEAR ACCUMULATOR FOR DELAY
172 80A4 8F01           DLY     1               ; DELAY 1 MS (1/4 BIT TIME)
173 80A6 CB04           ST      FLAG(P3)        ; RESET LATCH
174 80A8 C400           LDI     0               ; INIT ACCUMULATOR FOR DELAY
175 80AA 8F02           DLY     2               ; DELAY PAST MIDDLE OF WINDOW
176 80AC 06             CSA                     ; COPY STATUS TO ACCUMULATOR
177 80AD D420           ANI     X'20            ; MASK
178 80AF 9802           JZ      ONE             ; IF ZERO, THEN BIT IS A "1"
179 80B1 9004           JMP     RESET
180 80B3 40     ONE:    LDE
181 80B4 DC80           ORI     X'80            ; ADD "1" BIT TO CHARACTER
182 80B6 01             XAE                     ; SAVE IN E REGISTER
183 80B7 CB04  RESET:   ST      FLAG(P3)        ; RESET LATCH
184 80B9 06             CSA                     ; COPY STATUS TO ACCUMULATOR
185 80BA D420           ANI     X'20            ; MASK
186 80BC 98F9           JZ      RESET           ; CHECK IF LATCH IS RESET
187 80BE C209  RETRN3:  LD      TEMP4(P2)       ; RESTORE P2
188 80C0 33             XPAL    P3
189 80C1 C208           LD      TEMP3(P2)
190 80C3 37             XPAH    P3
191 80C4 3D             XPPC    P1              ; RETURN
192 80C5 90C9           JMP     GETBIT
193
194
                        PAGE 'DATA WRITE ROUTINES'
195
196
197             ; SEND 4 SECONDS OF "0" (ABOUT 1000) TO ALLOW FOR
198             ; TAPE TO SETTLE ON PLAY BACK AND ACT AS LEADER
199
200             ; OUTPUT OPERATION OF LED INDICATORS:
```

Figure 2C4–5 (Continued)

```
201
202                     SEARCH LED ON WHEN LEADER COMPLETE
203                     SEARCH LED OFF WHEN TRANSMISSION COMPLETE
204                     END OF TAPE LED ON WHEN TRANSMISSION COMPLETE
205
206
207 80C7 C400  INIT:   LDI     L(RAM)          ; PLACE RAM POINTER IN P2
208 80C9 32             XPAL    P2                    (LOWER)
209 80CA C482           LDI     H(RAM)
210 80CC 36             XPAH    P2              ;     (UPPER)
211 80CD C400           LDI     L(PERIPH)       ; PLACE PERIPHERAL ADDRESS
212 80CF 33             XPAL    P3              ; IN P3
213 80D0 C483           LDI     H(PERIPH)
214 80D2 37             XPAH    P3
215 80D3 C400  COMP:    LDI     0               ; CLEAR ACCUMULATOR
216 80D5 02             CCL                     ; CLEAR CARRY/LINK FLAG
217 80D6 FA0B           CAD     WDCNTL(P2)      ; FORM 1'S COMP OF LOWER COUNT
218 80D8 CA0B           ST      WDCNTL(P2)
219 80DA C400           LDI     0               ; CLEAR ACCUMULATOR
220 80DC FA0A           CAD     WDCNTU(P2)      ; FORM 1'S COMP OF UPPER COUNT
221 80DE CA0A           ST      WDCNTU(P2)
222 80E0 CB03           ST      SRCHOF(P3)      ; TURN OFF SEARCH LED
223 80E2 CB01           ST      EOTOFF(P3)      ; TURN OFF END OF TAPE LED
224 80E4 C408  SNDLDR:  LDI     8               ; SET OUTER COUNTER
225 80E6 CA01           ST      CNTL(P2)
226 80E8 C480  CNT1.    LDI     X'80            ; SET INNER COUNTER
227 80EA CA00           ST      CNTU(P2)
228 80EC CB04  CNT2.    ST      FLAG(P3)        ; PULSE WRITE FLAG
229 80EE C400           LDI     0               ; CLEAR ACCUMULATOR
230 80F0 8F04           DLY     4               ; DELAY 1 BIT TIME
231 80F2 BA00           DLD     CNTU(P2)        ; DECREMENT INNER COUNTER
232 80F4 9CF6           JNZ     CNT2            ; CHECK FOR ZERO
233 80F6 BA01           DLD     CNTL(P2)        ; DECREMENT OUTER COUNTER
234 80F8 94EE           JP      CNT1            ; CHECK FOR LESS THEN ZERO
235 80FA CB02           ST      SRCHON(P3)      ; TURN ON SEARCH LED
236
237
238               BLOCK TRANSFER ROUTINE.  SENDS BLOCK OF DATA TO CASETTE
239
240               THE FOLLOWING ADDRESSES MUST BE LOADED BY USER BEFORE
241               EXECUTING THE WRITE PROGRAM:
242
243               X'8203 -- UPPER 8 BITS OF PROGRAM ADDRESS
244               X'8204 -- LOWER 8 BITS OF PROGRAM ADDRESS
245               X'820A -- UPPER 8 BITS OF PROGRAM LENGTH
246               X'820B -- LOWER 8 BITS OF PROGRAM LENGTH
247             ; X'820C -- UPPER 8 BITS OF TRANSFER ADDRESS (ENTRY POINT)
248             ; X'820D -- LOWER 8 BITS OF TRANSFER ADDRESS
249
250
251 80FC C400  BLOCK:  LDI     0               ; CLEAR ACCUMULATOR
252 80FE CA02           ST      CKSUM(P2)       ; INITIALIZE CHECKSUM COUNTER
253 8100 C481           LDI     H(WRITE)        ; PLACE ADDRESS OF WRITE IN P1
254 8102 35             XPAH    P1
255 8103 C444           LDI     L(WRITE)-1
```

Figure 2C4–5 (Continued)

```
256 8105 31             XPAL    P1
257 8106 C4A5           LDI     X'A5            ; LOAD ACCUMULATOR WITH ID
258 8108 3D             XPPC    P1              ; WRITE ID ON TAPE
259 8109 C204           LD      STARTL(P2)      ; GET STARTING ADDRESS
260 810B 3D             XPPC    P1              ; WRITE ONTO TAPE
261 810C C203           LD      STARTU(P2)
262 810E 3D             XPPC    P1
263 810F C20D           LD      JUMPL(P2)         GET TRANSFER ADDRESS
264 8111 3D             XPPC    P1
265 8112 C20C           LD      JUMPU(P2)
266 8114 3D             XPPC    P1
267 8115 C20B           LD      WDCNTL(P2)      ; GET LENGTH
268 8117 3D             XPPC    P1
269 8118 C20A           LD      WDCNTU(P2)
270 811A 3D             XPPC    P1
271 811B C204  GETBYT:  LD      STARTL(P2)      ; PLACE CURRENT ADDRESS IN P1
272 811D 31             XPAL    P1
273 811E C203           LD      STARTU(P2)
274 8120 35             XPAH    P1
275 8121 C501           LD      @1(P1)            GET CHARACTER THROUGH
276 8123 01             XAE                       POINTER AND SAVE IN E REG.
277 8124 C444           LDI     L(WRITE)-1      ; GET ADDRESS OF WRITE AND
278 8126 31             XPAL    P1                SAVE CURRENT CONTENTS OF P1
279 8127 CA04           ST      STARTL(P2)
280 8129 C481           LDI     H(WRITE)
281 812B 35             XPAH    P1
282 812C CA03           ST      STARTU(P2)
283 812E 40             LDE
284 812F F202           ADD     CKSUM(P2)         UPDATE CHECKSUM
285 8131 CA02           ST      CKSUM(P2)
286 8133 40             LDE                     ; PLACE CHARACTER IN ACC.
287 8134 3D             XPPC    P1                SEND CHARACTER
288 8135 AA0B           ILD     WDCNTL(P2)        INCREMENT WORD COUNTER
289 8137 9CE2           JNZ     GETBYT            CHECK FOR ZERO
290 8139 AA0A           ILD     WDCNTU(P2)
291 813B 9CDE           JNZ     GETBYT
292 813D C202           LD      CKSUM(P2)       ; SEND CHECKSUM TO TAPE
293 813F 3D             XPPC    P1
294 8140 CB03           ST      SRCHOF(P3)      ; TURN OFF SEARCH LED
295 8142 CB00           ST      EOTON(P3)       ; TURN ON END OF TAPE LED
296 8144 00             HALT                    ; HALT WHEN FINISHED
297
298
299             DATA WRITE ROUTINE.  WRITES 1 8-BIT CHARACTER ON TAPE
300
301
302 8145 01    WRITE:   XAE                       SAVE CHARACTER IN E REG.
303 8146 C408           LDI     8                 SET BIT COUNT
304 8148 CA05           ST      BITCNT(P2)
305 814A 40    MASK:    LDE
306 814B D401           ANI     1               ; MASK
307 814D 9C08           JNZ     SEND1           ; CHECK IF BIT "0" OR "1"
308 814F C400           LDI     0               ; CLEAR ACCUMULATOR FOR DELAY
309 8151 CB04  SEND0:   ST      FLAG(P3)        ; PULSE WRITE FLAG
310 8153 8F04           DLY     4               ; DELAY 1 BIT TIME ( 4 MS )
```

Figure 2C4–5 (Continued)

```
311 8155 900C            JMP     SHIFT
312 8157 C400  SEND1:    LDI     0
313 8159 CB04            ST      FLAG(P3)          ; PULSE WRITE FLAG
314 815B 8F02            DLY     2                 ; DELAY TO MIDDLE OF WINDOW
315 815D CB04            ST      FLAG(P3)          ; PULSE WRITE FLAG
316 815F C400            LDI     0                 ; CLEAR ACC.  FOR DELAY
317 8161 8F02            DLY     2                 ; DELAY TO END OF WINDOW
318
319 8163 19    SHIFT:    SIO                       ; SHIFT E REGISTER
320 8164 BA05            DLD     BITCNT(P2)        ; DECREMENT BIT COUNTER
321 8166 9802            JZ      RETRN1            ; CHECK FOR ZERO
322 8168 90E0            JMP     MASK              ; SEND NEXT BIT
323 816A 3D    RETRN1:   XPPC    P1                ; RETURN
324 816B 90D8            JMP     WRITE
325
326       8000             END     BOOT
```

BITCNT	0005	BLOCK	80FC *	BOOT	8000
BOOTIN	803E	CKSA	809B	CKSUM	0002
CLOCK	80A2	CNT1	80E8	CNT2	80EC
CNTL	0001	CNTU	0000	COMP	80D3 *
EOTOFF	0001	EOTON	0000	EXECPR	805B
FLAG	0004	GETBIT	8090	GETBYT	811B
INIT	80C7 *	JUMPL	000D	JUMPU	000C
LOCID	801E	LOOP	807F	MASK	814A
ONE	80B3	P1	0001	P2	0002
P3	0003	PERIPH	8300	RAM	8200
RECV	806E	RESET	80B7	RETRN1	816A
RETRN2	8086	RETRN3	80BE *	SEND0	8151 *
SEND1	8157	SETPNT	8026	SHIFT	8163
SNDLDR	80E4 *	SRCHOF	0003	SRCHON	0002
STARTL	0004	STARTU	0003	TEMP1	0006
TEMP2	0007	TEMP3	0008	TEMP4	0009
WDCNTL	000B	WDCNTU	000A	WRITE	8145

```
NO ERROR LINES
SOURCE CHECKSUM=C694
```

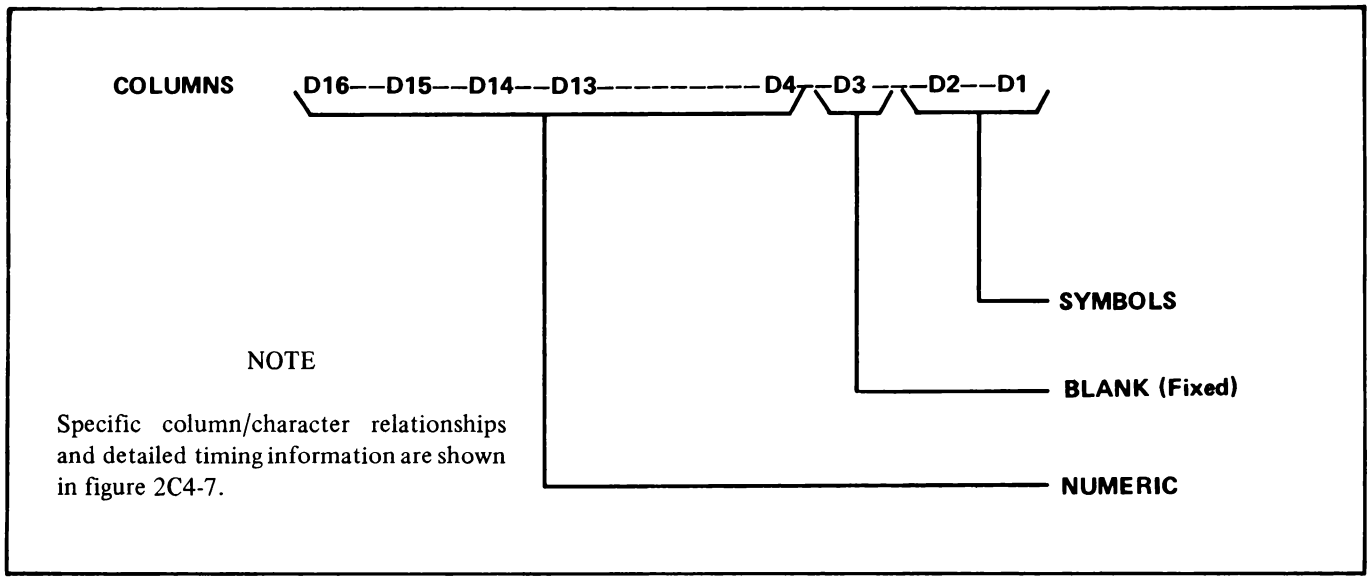
NS10567

Figure 2C4–5 (Concluded)

INTERFACING SC/MP WITH A SEIKO PRINTER

General Description

Figure 2C4-6 shows how the SC/MP microprocessor can be interfaced with the Model 310 Seiko Digital Printer; this particular printer is small and compact, and is widely used in applications where simplicity and economy are prime factors. The printer requires a single supply voltage (17V) and can operate at speeds in the range of two to five lines per second – with a choice of two colors. The Model 310 provides a 16-column print format with any one of 12 characters selected per column. The columns generally can be designated as follows.



Functional operation of the printer is described in documents furnished by the manufacturer (Shinshu Seiki Co., Ltd. of Japan); however, as an aid to the user, an operational summary is included here. When a print command (figure 2C4-7) is received from SC/MP, the motor-drive signal is driven low; accordingly, the printer motor is activated and the main shaft begins to rotate. The 16 print wheels (D1 through D16) are mechanically linked to the shaft and the position of each wheel is detected by a photo-diode arrangement. A timing pulse (T_0, T_1, T_2--------T_{11}) is generated for each digit or symbol position of the print wheels and, for any given print cycle, the timing pulses select the digit (D1) position for each wheel. During the T_4 to T_8 interval, color information is transmitted to select the color of tape to be printed. When the position of the wheel corresponds to the selected digit (or symbol) for that column, the print wheel is stopped and mechanically latched; thus, at the trailing edge of timing pulse T_{11}, all the print wheels are locked in position. When the platten print signal goes high, the selected characters for each column are transferred to the paper and, then, the paper is advanced. After completion of the print cycle, the motor drive signal is terminated and each print wheel returns to the initial blank (B) position.

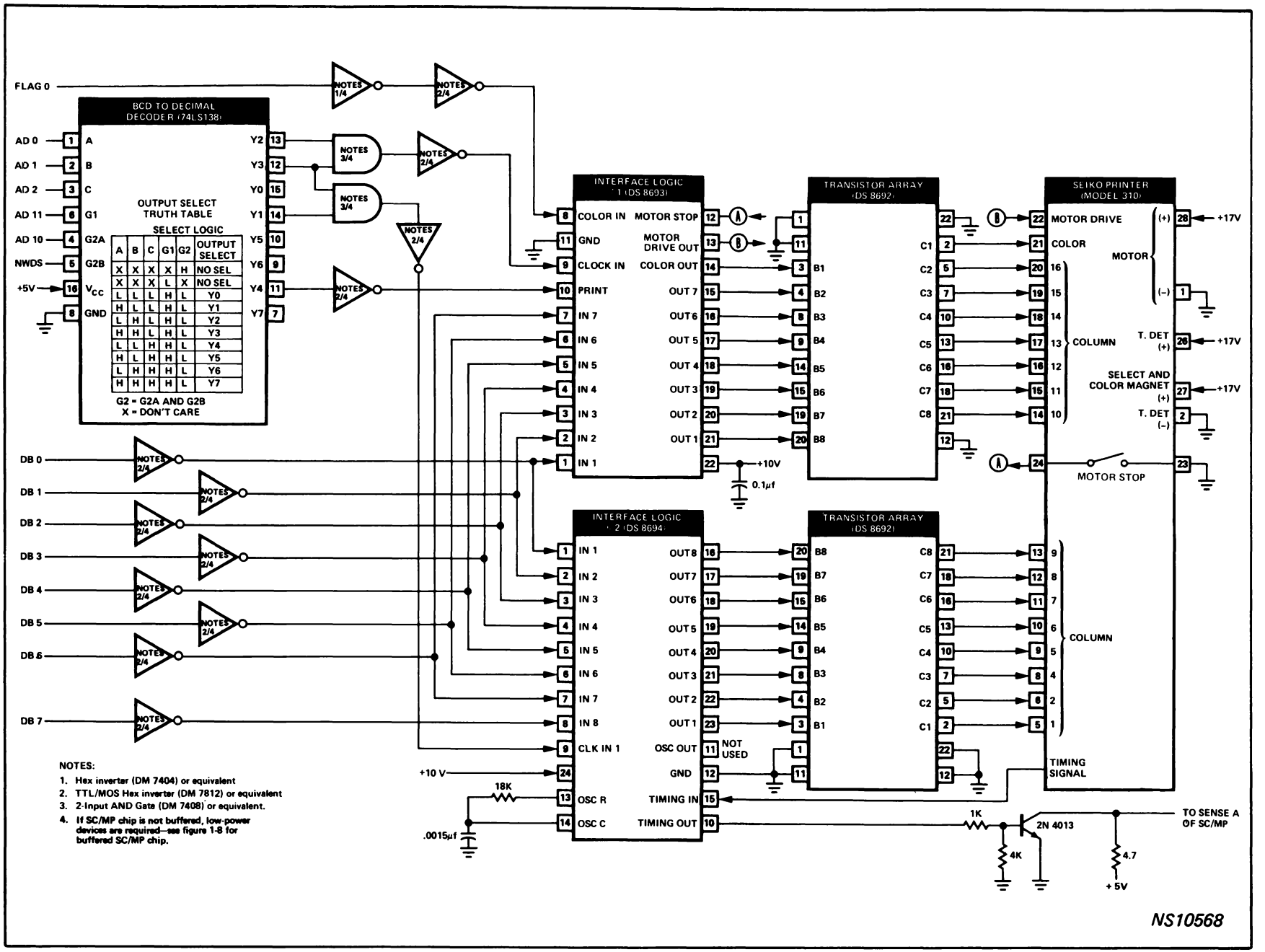
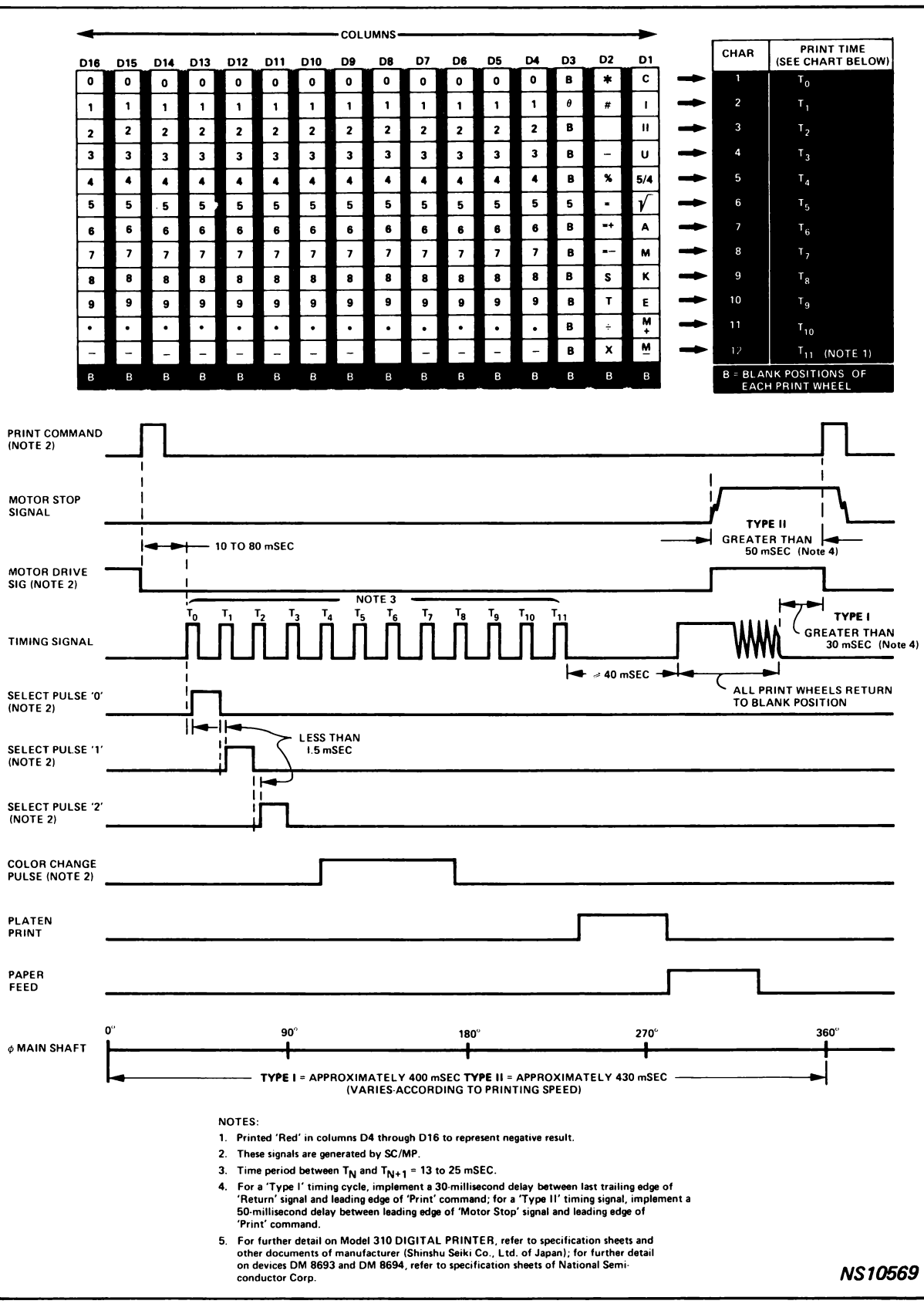


Figure 2C4–6. SC/MP Interfaced with Seiko Digital Printer



NOTES:

1. Printed 'Red' in columns D4 through D16 to represent negative result.
2. These signals are generated by SC/MP.
3. Time period between T_N and T_{N+1} = 13 to 25 mSEC.
4. For a 'Type I' timing cycle, implement a 30-millisecond delay between last trailing edge of 'Return' signal and leading edge of 'Print' command; for a 'Type II' timing signal, implement a 50-millisecond delay between leading edge of 'Motor Stop' signal and leading edge of 'Print' command.
5. For further detail on Model 310 DIGITAL PRINTER, refer to specification sheets and other documents of manufacturer (Shinshu Seiki Co., Ltd. of Japan); for further detail on devices DM 8693 and DM 8694, refer to specification sheets of National Semiconductor Corp.

NS10569

Figure 2C4–7. Column/Character Relationships and Timing for One Print Cycle

The SC/MP-to-printer interface is implemented via a special-purpose chip set that includes interface logic #1 (DS 8693), interface logic #2 (DS 8694), and two transistor arrays (DS 8692). The DS 8693 device contains the interface logic for the color solenoid driver, the motor driver, and seven of the column/character select solenoid drivers; the DS 8694 chip contains the interface logic for eight column/character solenoid drivers plus the clock oscillator and timing-signal buffer. Each transistor array contains eight common-emitter output circuits – each circuit features active pull down and each can sink up to 350 milliamperes of current. Address decoding for the printer interface is performed by a BCD-to-decimal decoder (DM 74LS138). Hexadecimal address X'0200 is assigned to access the printer; address assignments for interface control are as follows:

Hex Address	Function	Remarks
0200	Printer Interface	
0201	Clock IN1	Used to load DS 8693 with print information for columns D10 through D16
0202	Clock IN2	Used to load DS 8694 with print information for columns D1 through D9
0203	Common Clock	Used to clear DS 8693 and DS 8694
0204	Print	Used to issue PRINT COMMAND

The printer program continuously monitors a data buffer that is maintained in RAM (figure 2C4-8). This buffer is filled by any appropriate input device (keyboard, tape, or other), and when filled, the program is executed to print a line consisting of 16 characters. As shown in figure 2C4-7, the 16 columns (each column corresponding to a print wheel) are divided into two column words – COLWORD 1 representing the characters to be printed for columns D1 through D9 (D3 is blank) and COLWORD 2 representing characters for the remaining columns (D10 through D16). The character codes for each column and the constants used to select a particular column are stored in ROM. When the characters stored in the data buffer for a particular column agree with those in the character code list, the print wheel for that column is mechanically latched; thus, at the end of the timing cycle (T_0 - - - - T_{11}), all 16 print wheels are locked in position and the line is printed. The following example shows the interrelationships between the columns, the characters, the timing pulses, and COLWORDS 1 and 2.

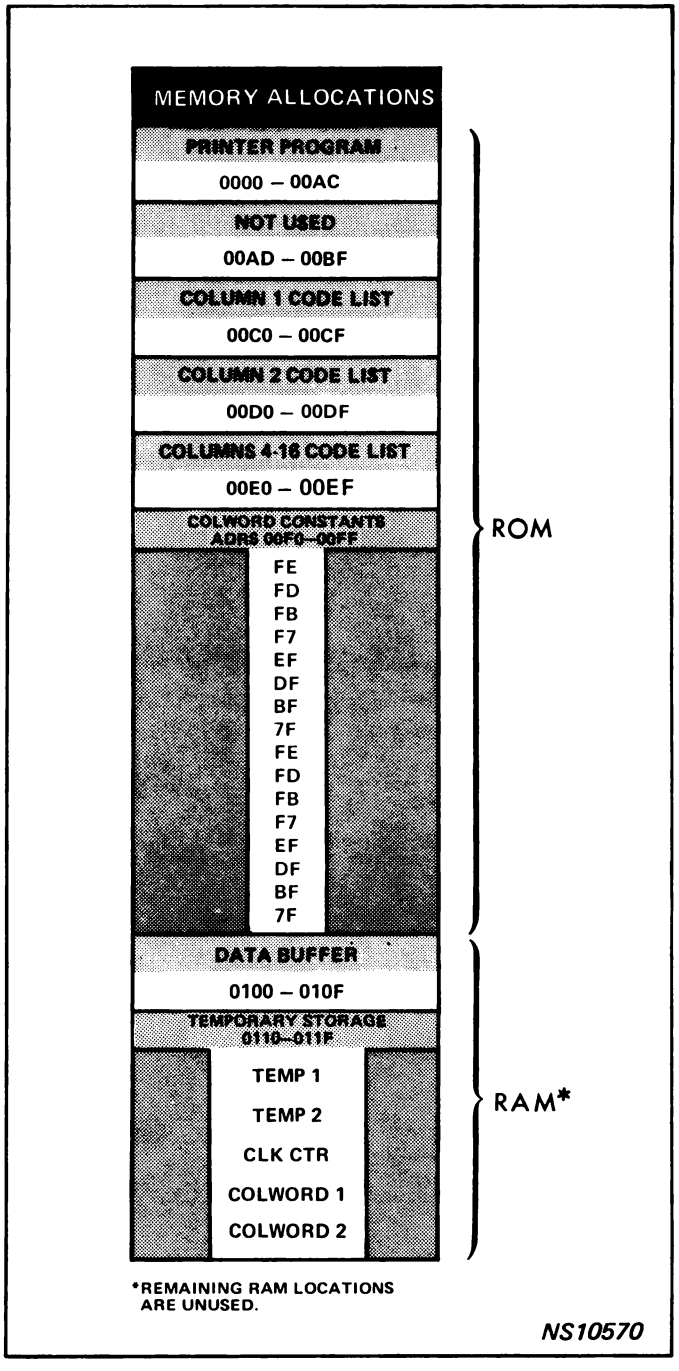


Figure 2C4–8. Memory Allocations for Printer Program

(NOTE: The TTL to MOS inverters in the interface devices require that the column drivers be driven with a logic '0' for selection; that is, the print wheels lock into position at a particular timing pulse if the COLWORD bit corresponding to that position is a logic '0'.)

After the line of data is printed, all print wheels are unlatched and return to the blank (B) position; the motor drive signals also are reset until the arrival of the next print command.

Software Considerations

The flowcharts (figure 2C4-9) and program listing (figure 2C4-10) shows how the SC/MP-to-printer interface can be software-controlled to provide the foregoing printing capabilities.

COLUMNS →	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1
CHARACTER →	9	8	7	6	5	4	3	2	1	0	.	6	6	--	*	M
	COLWORD 2							COLWORD 1								
	BITS							BITS								
	6	5	4	3	2	1	0	7	6	5	4	3	2		1	0
TIMING SIGNAL																
T_0 →	1	1	1	1	1	1	1	1	1	0	1	1	1	--	0	1
T_1 →	1	1	1	1	1	1	1	1	0	1	1	1	1	--	1	1
T_2 →	1	1	1	1	1	1	1	0	1	1	1	1	1	--	1	1
T_3 →	1	1	1	1	1	1	0	1	1	1	1	1	1	--	1	1
T_4 →	1	1	1	1	1	0	1	1	1	1	1	1	1	--	1	1
T_5 →	1	1	1	1	0	1	1	1	1	1	1	1	1	--	1	1
T_6 →	1	1	1	0	1	1	1	1	1	1	1	0	0	--	1	1
T_7 →	1	1	0	1	1	1	1	1	1	1	1	1	1	--	1	1
T_8 →	1	0	1	1	1	1	1	1	1	1	1	1	1	--	1	1
T_9 →	0	1	1	1	1	1	1	1	1	1	1	1	1	--	1	1
T_{10} →	1	1	1	1	1	1	1	1	1	1	0	1	1	--	1	1
T_{11} →	1	1	1	1	1	1	1	1	1	1	1	1	1	--	1	0

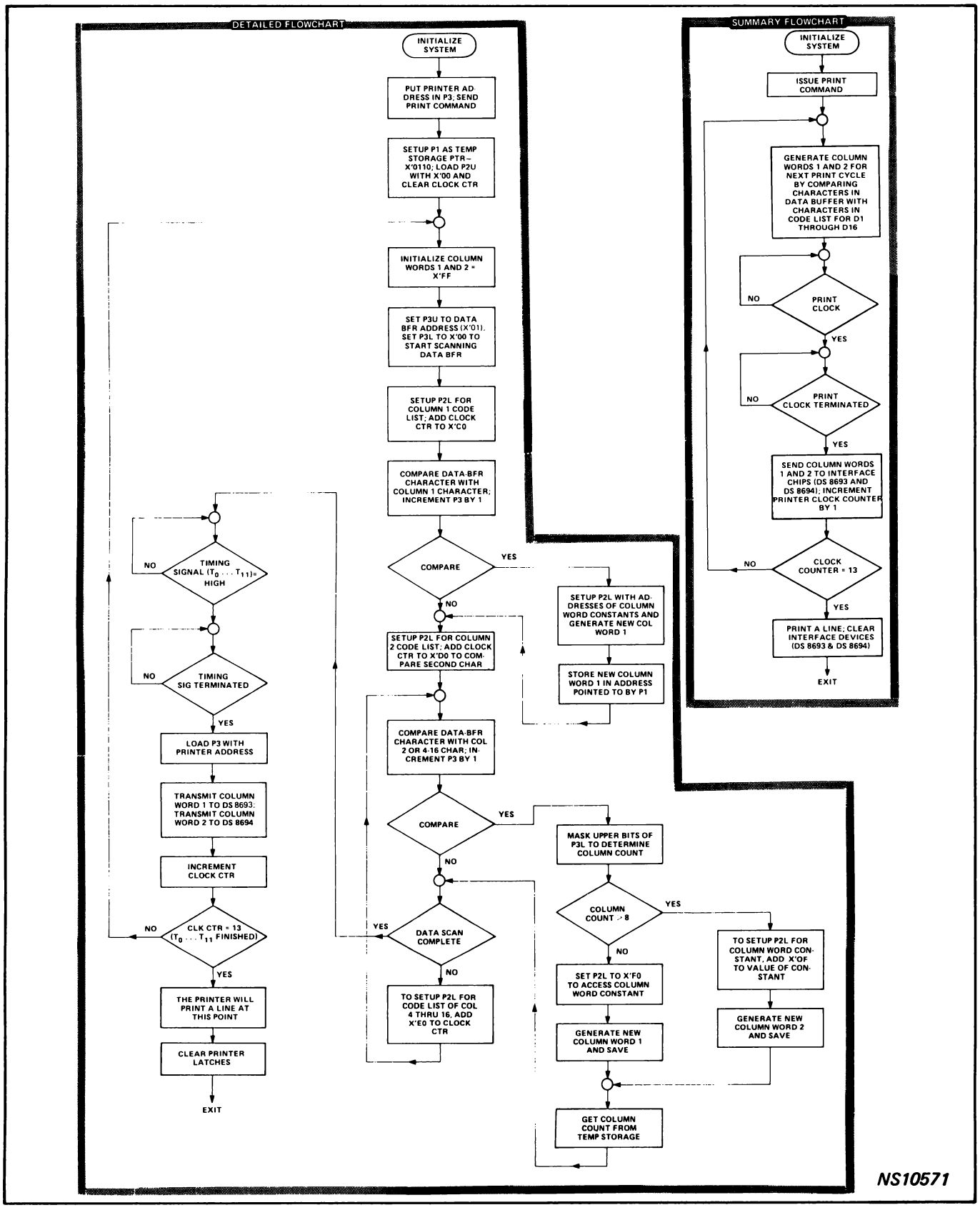


Figure 2C4–9. Summary and Detailed Flowchart for SC/MP-to-Printer Interface

```
 1                          .TITLE  SCMP, 'SEIKO PRINTER PRGM'
 2
 3        0001  P1         =       1
 4        0002  P2         =       2
 5        0003  P3         =       3
 6        0001  CLK1       =       1
 7        0002  CLK2       =       2
 8        0003  CLK3       =       3
 9        0004  PRINT      =       4
10        0200  SEIKO      =       0200
11        0001  TEMP1      =       1
12        0002  TEMP2      =       2
13        0004  CLKCTR     =       4
14        0005  COLWR1     =       5
15        0006  COLWR2     =       6
16
17 0000 08                 NOP
18 0001 C400   PRNT:       LDI     L(SEIKO)          ;SET UP POINTER ADRS.
19 0003 33                 XPAL    P3
20 0004 C402               LDI     H(SEIKO)
21 0006 37                 XPAH    P3
22 0007 CB04               ST      PRINT(P3)         ;START PRINTER
23 0009 C401               LDI     1                 ;SET UP HIGHER MEMORY ADRS.
24 000B 35                 XPAH    P1
25 000C C400               LDI     0
26 000E 36                 XPAH    P2
27 000F C410               LDI     010
28 0011 31                 XPAL    P1                ;SET UP TEMP. STORAGE ADRS.
29 0012 C400               LDI     0
30 0014 C904               ST      CLKCTR(P1)        ;CLR CLK COUNTER
31
32 0016 C4FF   CONTPRT:    LDI     0FF
33 0018 C905               ST      COLWR1(P1)        ;CLR COL WORD1
34 001A C906               ST      COLWR2(P1)        ;CLR COL WORD2
35 001C C401               LDI     1                 ;SET UP DATA BFR ADRS.
36 001E 37                 XPAH    P3
37 001F C400               LDI     0
38 0021 33                 XPAL    P3
39 0022 C104               LD      CLKCTR(P1)        ;SET UP POINTER FOR
40 0024 F4C0               ADI     0C0               ;COL1 CODE LIST
41 0026 32                 XPAL    P2
42 0027 C701               LD      @1(P3)            ;BEGIN SCAN FOR DATA COMPR
43 0029 E200               XOR     (P2)
44 002B 9827               JZ      COMPR1
45 002D C104   RETCOMP:    LD      CLKCTR(P1)        ;SET UP POINTER FOR CODE LIST
46
47 002F F4D0               ADI     0D0               ;SELECT COL2 CODE LIST
48 0031 32                 XPAL    P2
49 0032 C701               LD      @1(P3)            ;NO PRINTING IN COL3
50 0034 E200   CONTSCN:    XOR     (P2)              ;CHECK IF DATA=CODE LIST CHAR
51 0036 9827               JZ      COMPR2
52 0038 33                 XPAL    P3
53 0039 C901               ST      TEMP1(P1)
54 003B D40F               ANI     0F                ;MASK UPPER 4 BITS OF P3
```

Figure 2C4–10. Program Listing for SC/MP-to-Printer Interface

```
 55 003D E40F  SCNCHK: XRI     0F              ;CHECK IF SCANNING COMPLT.
 56 003F 980C          JZ      TEST            ;JMP TO WAIT FOR PRTR CLK
 57 0041 C101          LD      TEMP1(P1)       ;LD DATA BFR ADRS.
 58 0043 33            XPAL    P3
 59 0044 C104          LD      CLKCTR(P1)
 60 0046 F4E0          ADI     0E0             ;LD COL4 TO 16 CODE ADRS.
 61 0048 32            XPAL    P2
 62 0049 C701          LD      @1(P3)
 63 004B 90E7          JMP     CONTSCN
 64
 65 004D 06    TEST:   CSA
 66 004E D410          ANI     010
 67 0050 98FB          JZ      TEST
 68 0052 9033          JMP     FNSHSCN
 69
 70 0054 C4F0  COMPR1: LDI     0F0             ;LD COL WORD CONST.  ADRS.
 71 0056 32            XPAL    P2
 72 0057 C200          LD      (P2)            ;COLWORD CONST TO ACU
 73
 74 0059 D105          AND     COLWR1(P1)      ;GENERATE COLWRD 1
 75 005B C905          ST      COLWR1(P1)      ;SAVE NEW COLWRD1
 76 005D 90CE          JMP     RETCOMP
 77
 78 005F 33    COMPR2: XPAL    P3
 79 0060 C901          ST      TEMP1(P1)       ;SAVE DATA BFR ADRS
 80 0062 D40F          ANI     0F              ;MASK UPPER BITS
 81 0064 C902          ST      TEMP2(P1)       ;SAVE COLMN COUNT
 82 0066 F4F7          ADI     -9
 83 0068 940F          JP      GENCOL2         ;COL COUNT>8
 84 006A C102          LD      TEMP2(P1)       ;GET COL COUNT
 85 006C F4F0          ADI     0F0             ;ADD ADRS FOR COLWRD CONST
 86 006E 32            XPAL    P2
 87 006F C2FF          LD      -1(P2)          ;COLWRD CONST TO ACU
 88 0071 D105          AND     COLWR1(P1)      ;GENERATE COLWRD1
 89 0073 C905          ST      COLWR1(P1)      ;SAVE NEW COLWRD1
 90 0075 C102  COL2RET:LD      TEMP2(P1)       ;LD COLMN COUNT
 91 0077 90C4          JMP     SCNCHK          ;GO TO SCAN COMPLT CHK RTN.
 92 0079 C102  GENCOL2:LD      TEMP2(P1)       ;GET COL COUNT
 93 007B 02            CCL
 94 007C F4F0          ADI     0F0             ;ADD ADRS FOR COLWRD CONST
 95 007E 32            XPAL    P2
 96 007F C2FF          LD      -1(P2)          ;COLWRD CONST TO ACU
 97 0081 D106          AND     COLWR2(P1)      ;GENERATE COLWRD2
 98 0083 C906          ST      COLWR2(P1)      ;SAVE NEW COLWRD2
 99 0085 90EE          JMP     COL2RET
100
101 0087 06    FNSHSCN:CSA                    ;XFR SENSEA TO ACU
102 0088 D410          ANI     010             ;CHECK IF SENSE A PRESENT
103 008A 9CFB          JNZ     FNSHSCN         ;LOOP BACK TO FINISH SCAN
104 008C C400          LDI     L(SEIKO)        ;GET SEIKO ADRS.
105 008E 33            XPAL    P3
106 008F C402          LDI     H(SEIKO)
107 0091 37            XPAH    P3
108 0092 C105          LD      COLWR1(P1)      ;GET COLWRD1
109 0094 CB02          ST      CLK2(P3)        ;XFR COLWRD1 TO PRINTER
```

Figure 2C4–10 (Continued)

```
110 0096 C106           LD      COLWR2(P1)      ;GET COLWRD2
111 0098 CB01           ST      CLK1(P3)        ;XFR COLWRD2 TO PRINTER
112 009A A904           ILD     CLKCTR(P1)      ;INCREMENT CLK COUNTER
113 009C E40D           XRI     13              ;CHECK CLK CTR = 13 ·
114 009E 9808           JZ      DONE
115 00A0 C400           LDI     0
116 00A2 37             XPAH    P3
117 00A3 C415           LDI     015
118 00A5 33             XPAL    P3
119 00A6 93FF           JMP     (P3)
120 00A8 C4FF  DONE:    LDI     0FF
121 00AA CB03           ST      CLK3(P3)        ;CLR PRTR LATCHES
122
123            EXIT:                            ;USER RETURN ROUTINE
124
125
126      00C0           .=0C0
127
128 00C0 43    COL1:    .BYTE   'C','I','1','U','5','6','A','M'
    00C1 49
    00C2 31
    00C3 55
    00C4 35
    00C5 36
    00C6 41
    00C7 4D
129 00C8 4B             .BYTE   'K','E','P','N'
    00C9 45
    00CA 50
    00CB 4E
130
131      00D0           .=0D0
132
133 00D0 2A    COL2:    .BYTE   '*','#','+','-','%','=','6','7'
    00D1 23
    00D2 2B
    00D3 2D
    00D4 25
    00D5 3D
    00D6 36
    00D7 37
134 00D8 53             .BYTE   'S','T',':','X'
    00D9 54
    00DA 3A
    00DB 58
135
136      00E0           .=0E0
137
138 00E0 30    COL4:    .BYTE   '0','1','2','3','4','5','6','7'
    00E1 31
    00E2 32
    00E3 33
    00E4 34
    00E5 35
    00E6 36
```

Figure 2C4-10 (Continued)

```
    00E7 37
139 00E8 38              .BYTE   '8','9','.','-'
    00E9 39
    00EA 2E
    00EB 2D
140
141      00F0            .=0F0
142
143 00F0 FE     CONST:   .BYTE   0FE,0FD,0FB,0F7,0EF,0DF,0BF,07F
    00F1 FD
    00F2 FB
    00F3 F7
    00F4 EF
    00F5 DF
    00F6 BF
    00F7 7F
144 00F8 FE              .BYTE   0FE,0FD,0FB,0F7,0EF,0DF,0BF,07F
    00F9 FD
    00FA FB
    00FB F7
    00FC EF
    00FD DF
    00FE BF
    00FF 7F
145
146      0001            .END    PRNT


CLK1    0001          CLK2    0002          CLK3    0003
CLKCTR  0004          COL1    00C0 *        COL2    00D0 *
COL2RE  0075          COL4    00E0 *        COLWR1  0005
COLWR2  0006          COMPR1  0054          COMPR2  005F
CONST   00F0 *        CONTPR  0016 *        CONTSC  0034
DONE    00A8          EXIT    00AC *        FNSHSC  0087
GENCOL  0079          P1      0001          P2      0002
P3      0003          PRINT   0004          PRNT    0001
RETCOM  002D          SCNCHK  003D          SEIKO   0200
TEMP1   0001          TEMP2   0002          TEST    004D

NO ERROR LINES
SOURCE CHECKSUM=4E54
```

NS10572

Figure 2C4–10 (Concluded)

APPENDIX A

CLOCK CONSIDERATIONS FOR SC/MP

GENERAL

The on-chip oscillator and timing generator of SC/MP can be controlled by any one of the following three methods:

- CAPACITOR – if timing is not a critical parameter
- CRYSTAL – if more precise timing is required
- EXTERNAL DRIVE – if application requires that SC/MP be synchronized with system clock

CAPACITOR TIMING

As shown in figure A-1, the capacitor is connected between X1 (pin 37) and X2 (pin 38) of the SC/MP chip. A non-polarized ceramic or silver mica capacitor with a working voltage that is equal to or greater than 25 volts is recommended. Lead length from the body of SC/MP to the body of the capacitor should not exceed 1.25 inches. When a capacitor is used, the frequency varies according to the capacitance as shown in figure A-2.

CRYSTAL TIMING

When a crystal is used, the on-chip oscillator of SC/MP operates at the resonant frequency of the crystal. Pin connections and lead lengths for a crystal are identical to those for a capacitor; however, as shown in figure A-1, a capacitor may be required to suppress crystal harmonics. The crystal should be hermetically sealed (HC type can) and should meet the following electrical specifications.

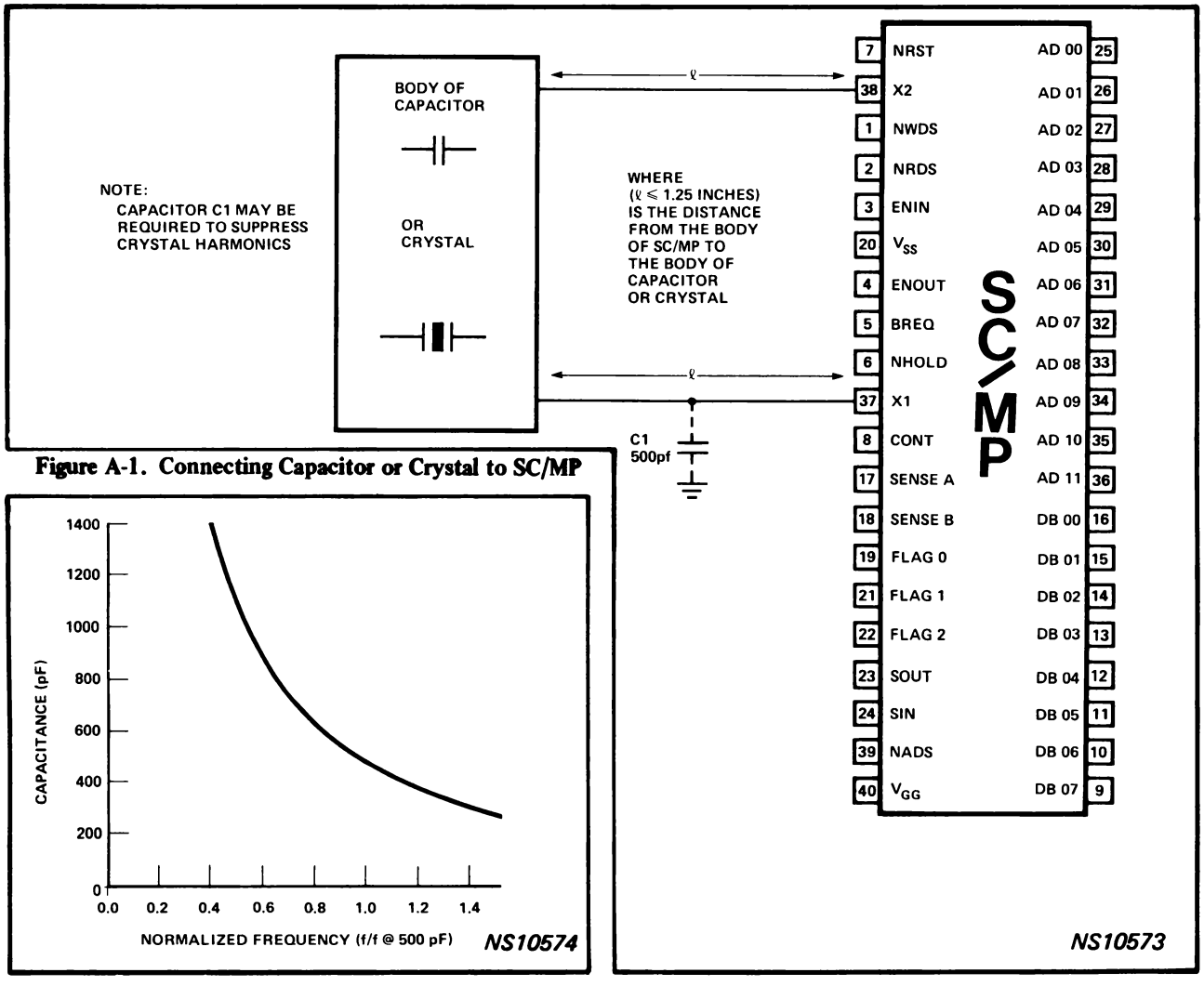


Figure A-1. Connecting Capacitor or Crystal to SC/MP

Figure A-2. Oscillator Frequency versus Capacitance

- Resonant frequency - - - - - - - - - - - - - - $\geqslant$ 900 kHz $\leqslant$ 1 MHz
- Series resistance at resonance - - - - - - - $\leqslant$ 600-ohms
- Load capacitance at resonance - - - - - - 20-to-30 pf

Suitable crystals can be obtained from several manufacturers; four such manufacturers are listed below.

- X-Tron Electronics, Hayward, California
- M-Tron Industries, Yankton, South Dakota
- Crystek Crystal Co., Ft. Myers, Florida
- JAN Crystals, Ft. Myers, Florida

USING AN EXTERNAL CLOCK

SC/MP can be synchronized with an external clock simply by connecting appropriate drive circuits to the X1/X2 inputs. A recommended method of implementing the external-clock circuit is shown in figure A-3a; the true and complemented "idealized" waveforms are shown in figure A-3b. Alternate methods of generating the X1/X2 input signals are shown in figure A-4.

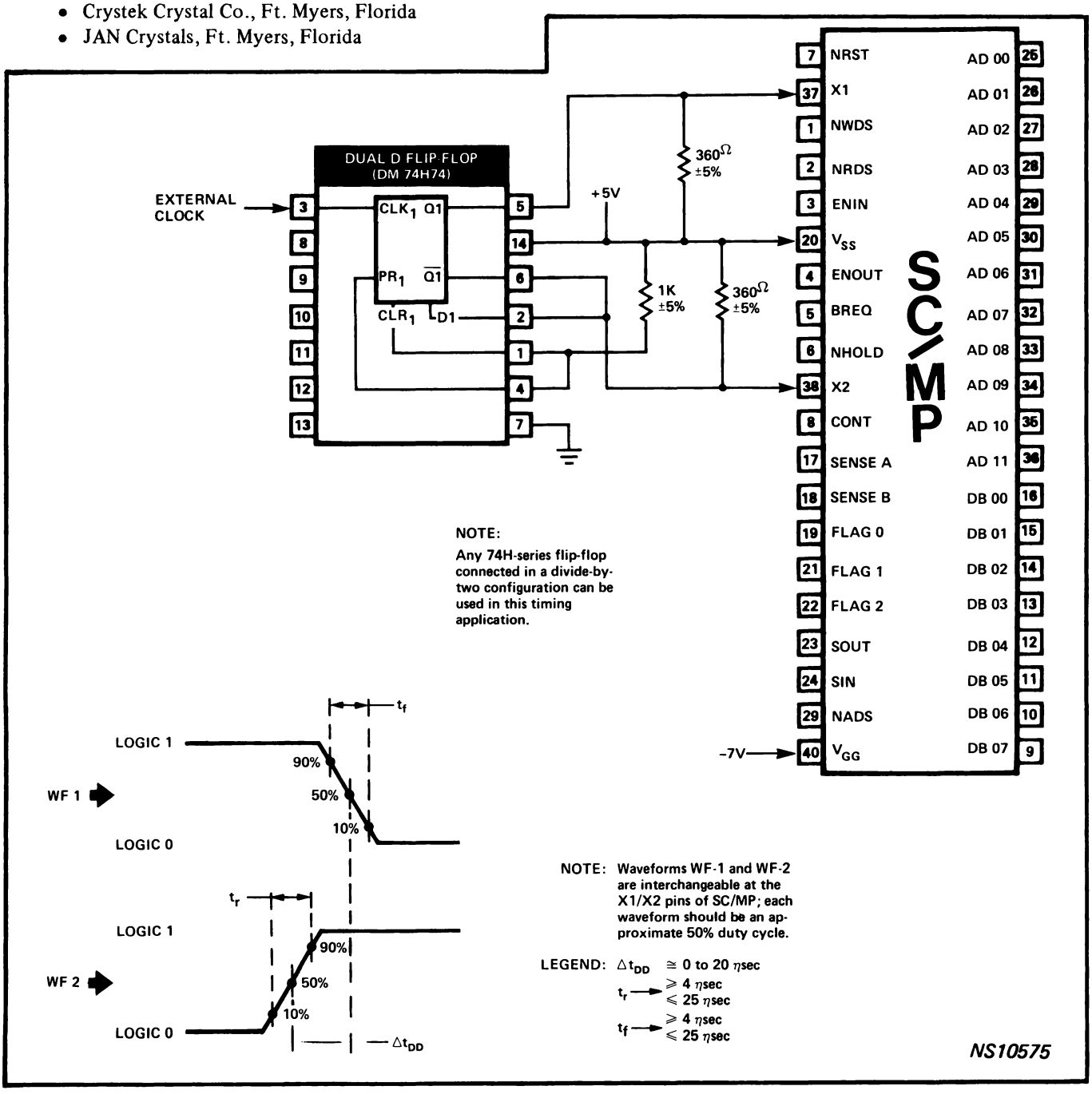


Figure A-3. Using External Clock for SC/MP Timing

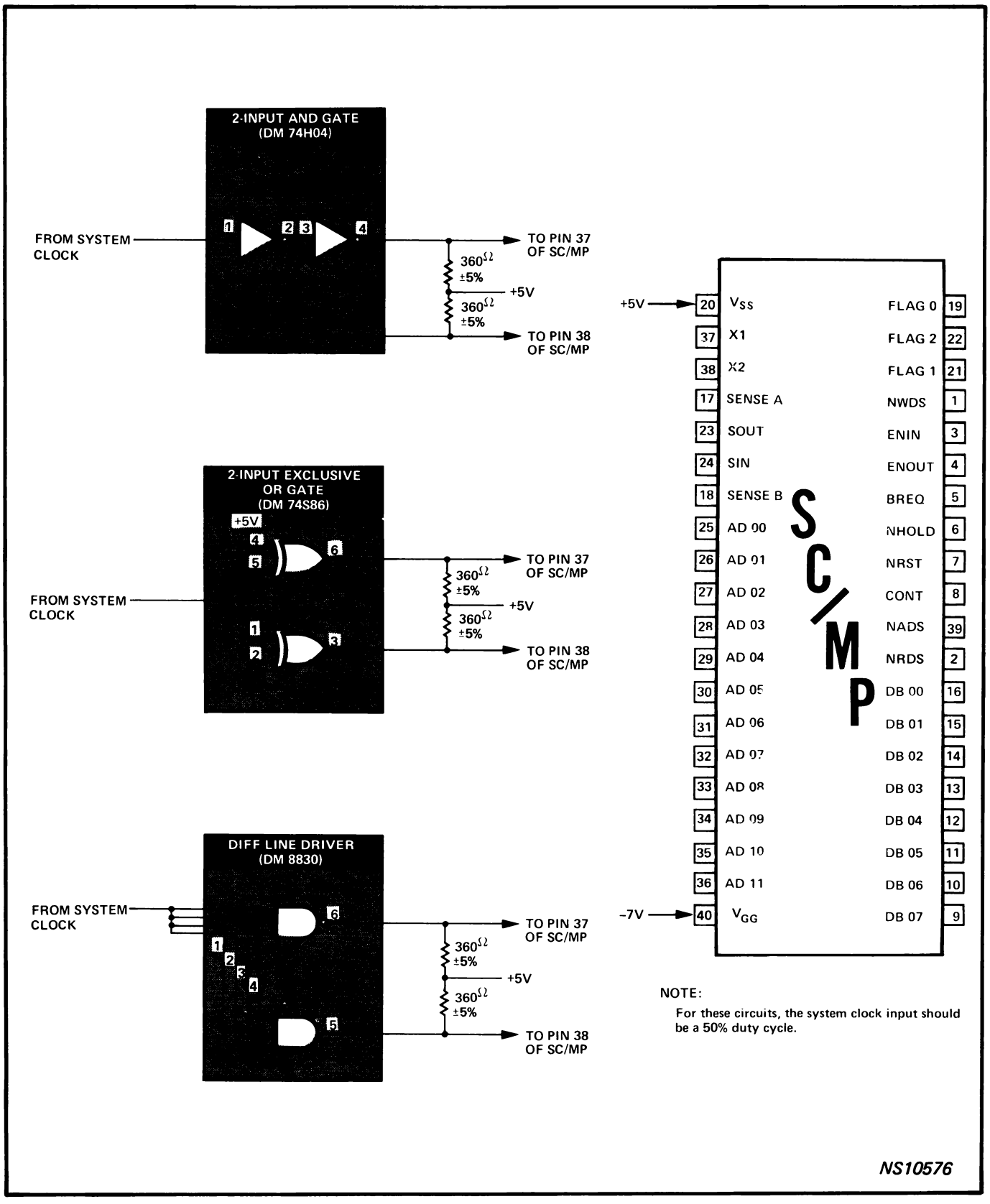


Figure A-4. Alternate Methods of Generating External Clock Signals

APPENDIX B

ADDRESS ASSIGNMENTS AND DECODING METHODS

INTRODUCTION

The addresses for memory and peripheral devices can be assigned and decoded in a number of different ways; some address assignments and decoding methods are shown and described in the sections that follow.

Nondecoded Peripheral Addressing

As previously indicated, 16 address lines (AD 00 through AD 15) are available; 12 of these lines (AD 00 through AD 11) are internally latched on the SC/MP chip; whereas the other 4 lines (AD 12 through AD 15) are output (at

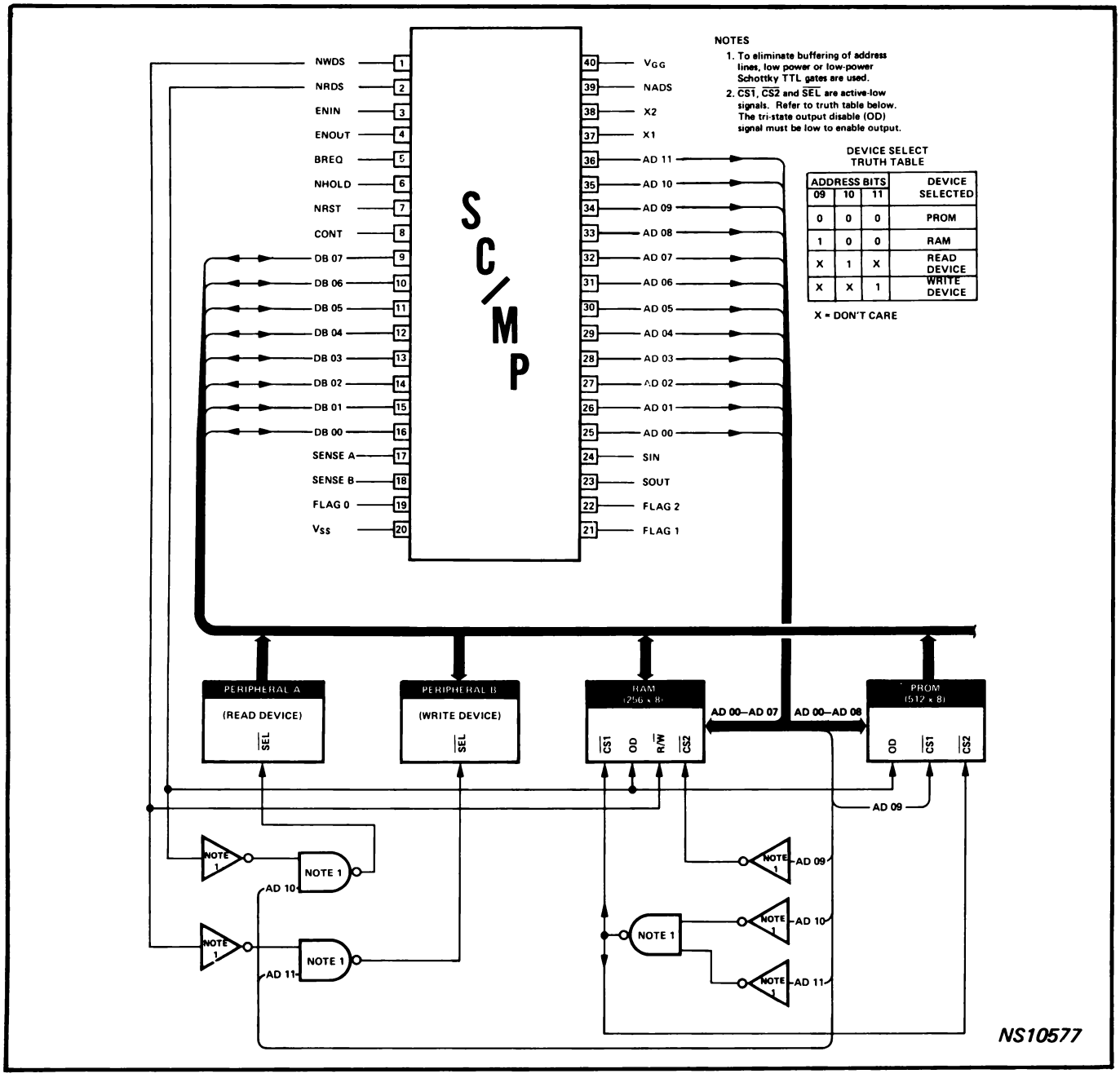


Figure B-1. Using External Logic and Spare Address Lines to Select RAM/PROM Memory or Input/Output Peripherals

NADS time) on the 8-bit input/output bus. Most applications do not require all 16 address lines for memory. For example, consider the system shown in figure B-1 consisting of 512-by-8 words of PROM, 256-by-8 words of RAM, a read device, and a write device. Nine address lines (AD 00-AD 08) are required to discretely identify each of the 512 bytes of PROM and only 8 lines (AD 00-AD 07) are required for RAM; thus, the 3 remaining address bits (AD 09/AD 10/AD 11) are free and can be used for device selection. The 'device select truth table' shows how this can be done.

When power is applied and SC/MP is initialized, all the address bits are low (set to '0') and PROM is selected; thus, program execution begins at PROM address X'0001. Observe that with AD 09 set to '0', RAM is not selected because $\overline{CS2}$ is high and neither the read nor the write devices can be selected because both $\overline{SEL}$ signals are high. For an application example, suppose the PROM program requires access to RAM, to the read device, and to the write device; the following series of instructions shows one way to implement the read and write functions.

```
READ FROM RAM:
                •
                •
                •
        LDI     02      ;WHEN TRANSFERRED TO HIGH POINTER BY
                        ;NEXT INSTRUCTION, TURNS ON BIT 9 TO
                        ;SELECT RAM
        XPAH    2       ;SET BIT 9 OF POINTER 2
        LDI     05      ;LOAD RAM ADDRESS 5 INTO ACCUMULATOR
        XPAL    2       ;PUT RAM ADDRESS IN LOW POINTER 2
        LD      (2)     ;LOAD DATA FROM RAM ADDRESS SPECIFIED IN
                        ;POINTER 2
                •
                •
                •
WRITE INTO RAM:
                •
                •
                •
        LDI     02      ;WHEN TRANSFERRED TO HIGH POINTER BY
                        ;NEXT INSTRUCTION, TURNS ON BIT 9 TO
                        ;SELECT RAM
        XPAH    2       ;SET BIT 9 OF POINTER 2
        LDI     X'10    ;LOAD RAM ADDRESS (DECIMAL 16) INTO
                        ;ACCUMULATOR
        XPAL    2       ;PUT RAM ADDRESS IN LOW POINTER 2
        LD      DATA    ;LOAD DATA TO BE STORED
        ST      (2)     ;STORE DATA IN RAM ADDRESS SPECIFIED BY
                        ;POINTER 2
DATA:   .BYTE   X'1F    ;DATA TO BE STORED
                •
                •
                •
READ FROM "READ DEVICE":
                •
                •
                •
        LDI     04      ;WHEN TRANSFERRED TO HIGH POINTER BY NEXT
                        ;INSTRUCTION, TURNS ON BIT 10 TO SELECT
                        ;"READ DEVICE"
        XPAH    1       ;SET BIT 10 OF POINTER 1
        LD      (1)     ;READ DATA FROM DEVICE (NOTE: LOW POINTER 1
                        ;IS NOT REQUIRED FOR THIS OPERATION)
                •
                •
                •
```

WRITE INTO "WRITE DEVICE":

```
                •
                •
                •
        LDI     08      ;WHEN TRANSFERRED TO HIGH POINTER BY NEXT
                        ;INSTRUCTION, TURNS ON BIT 11 TO SELECT
                        ;"WRITE DEVICE"
        XPAH    1       ;SET BIT 11 OF POINTER 1
        LD      DATA    ;LOAD DATA TO BE WRITTEN
        ST      (1)     ;WRITE DATA
DATA:   .BYTE   X'AA    ;DATA TO BE WRITTEN
                •
                •
                •
```

READ AND/OR WRITE:

```
        LDI     OC      ;WHEN TRANSFERRED TO HIGH POINTER BY
                        ;NEXT INSTRUCTION, TURNS ON BITS 10 & 11
                        ;TO SELECT READ AND WRITE DEVICE
        XPAH    1       ;SET BITS 10 & 11 OF PTR 1
        LD      (1)     ;READ DATA FROM READ DEVICE
        ST      (1)     ;WRITE DATA THAT WAS PREVIOUSLY READ
```

Observe that the read and write peripherals must be strobed by NRDS or NWDS; otherwise, the selected peripheral would input or output data as soon as the address is valid, and, at the same time, SC/MP would output address and status information. By strobing the chip selects, reading or writing of data is delayed until the "address" and "status" outputs from SC/MP are completed. Strobes are not required in the address logic of the RAM or PROM since the on-chip output-enable (OD) signal provides this function.

Decoded Addressing of Peripherals

In many applications, the addressing requirements of the system exceed those shown in figure B-1. For these systems, discrete selection logic is expensive, cumbersome, and complicates the software; thus, some form of address decoding is preferred. Figure B-2 shows one way of implementing a simple decoding scheme. Here, address bits 10 and 11 are decoded by one half of the 74LS155 to yield four output select signals; address bit 9 is used to select RAM or PROM. In figure B-2, all chip selects are strobed via the decoder; thus, access time of the memory chip is an important consideration. If the access time is greater than the strobe width, bipolar memories can be used or the RAM/PROM devices can be selected as shown in figure B-1—via the external logic and the latched address lines.

In figures B-1 and B-2, three latched address lines are always available since memory is arbitrarily restricted to 512 bytes. Some applications require 4K of memory (or more) and, in this case, there are no spare address lines for device selection; figure B-3 shows how address bits 12-15 can be used to implement a system of this type. The 4 address bits (AD 12-AD 15) are output on the data bus at "address/status time," and the leading edge of the address strobe (NADS) latches these bits into the DM85L51 chip. The binary address code then is inputted to the 4-by-16 decoder, which selects 1 of 16 peripheral input/output devices in accordance with the decoding select logic. In figure B-3, the 16 peripherals can be any combination of read, write, and read/write devices whose input/output characteristics are compatible with SC/MP. Observe that the decoder is strobed; thus, access time of all peripherals must be less than the strobe widths—refer to figure 1-4 for timing parameters.

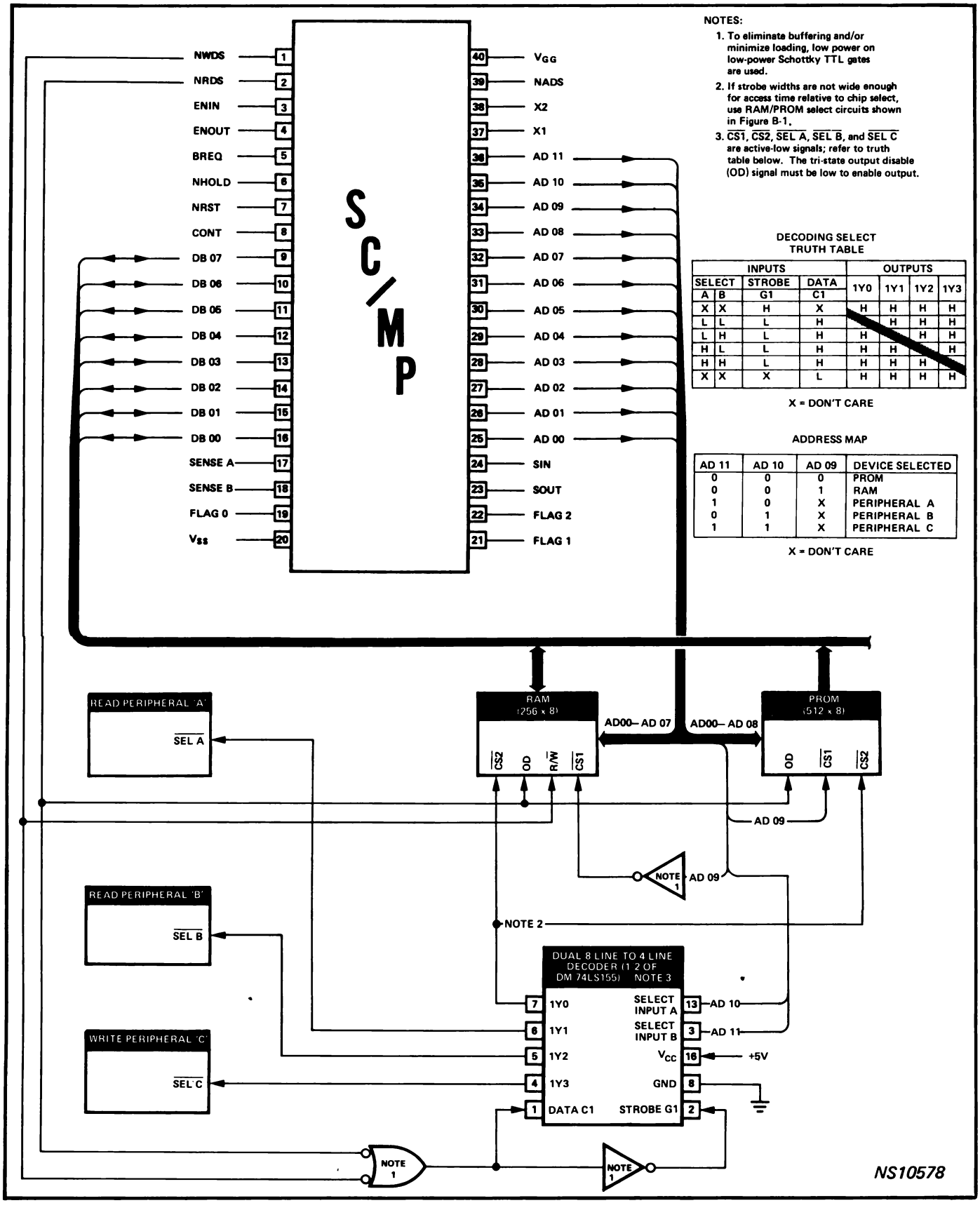


Figure B-2. Using 2-by-4 Decoder to Select Memory and Input/Output Peripherals

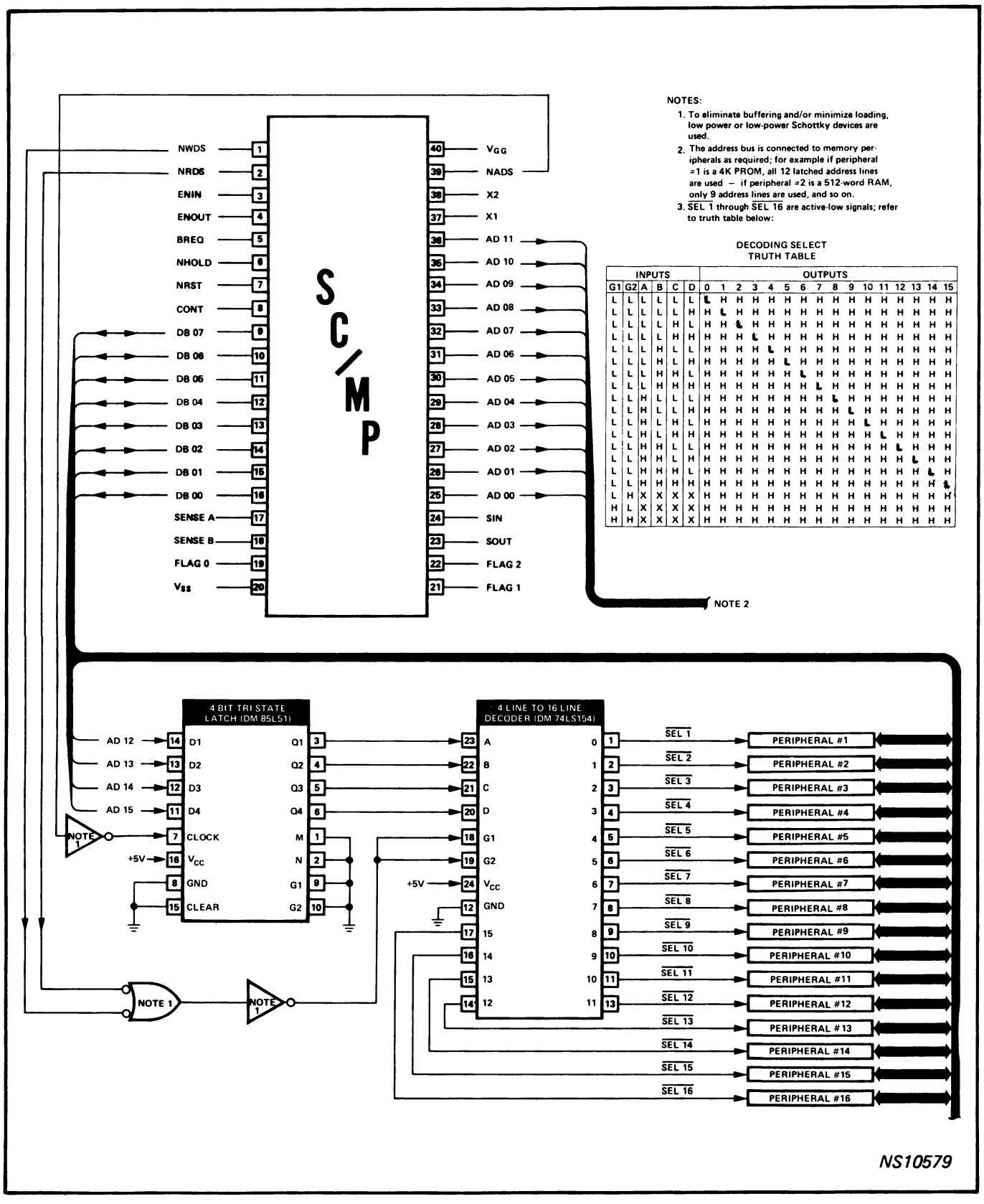


Figure B-3. Using Address Bits 12-15 and 4- by-16 Decoder to Select Any One of 16 Input/Output Peripherals

Figure B-4 shows how address decodes can be enabled via the write strobe (NWDS) and the read strobe (NRDS). Two decoders are used—one to generate eight write strobes and the other to generate eight read strobes. In systems that require multiple read/write peripherals, this decoding method is simple, easy to implement, and relatively inexpensive.

Another method of address decoding is shown in figure B-5. Here, each peripheral is selected by a 6-bit unified-bus comparator, and since the comparators have on-chip latches and are low-power devices (typically, a 15-microampere load), they are well suited to applications where physical space and low-power consumption are prime considerations. The selection code for each peripheral is designated by the user and is hardwired at pins 2, 4, 6, 10, 12, and 14 of the comparator; when the output address of SC/MP agrees with the preset code (T1 = B1, T2 = B2, and so on), the device is selected. For example, with the select codes shown, PROM is selected when power is applied and the system is initialized; thus, any program beginning at address X'0001 in PROM will be executed. To select RAM, the preset code can be altered as shown by changing 'T6' (AD 15) from a '0' to a '1.'

If a 4K memory is used in figure B-5, address lines AD 10 and AD 11 are unavailable for device selection; thus, these pins (B1 and B2) and their preset counterparts (T1 and T2) can be grounded or connected to +5V—whichever is most convenient. Observe that each peripheral device (whether read or write) is strobed by both NRDS and NWDS; this is necessary because the outputs are latched and must be cleared at the beginning of each input/output cycle.

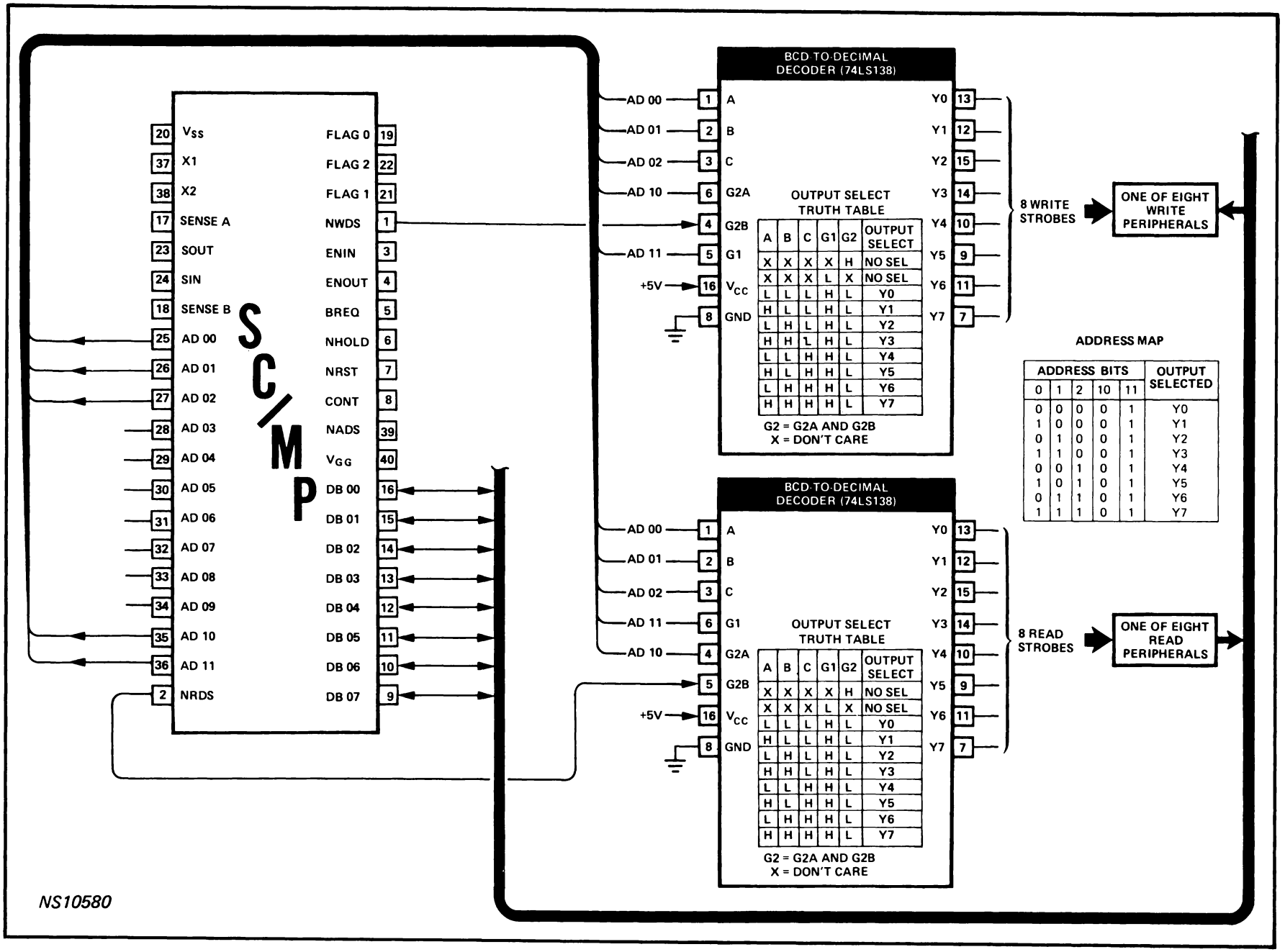


Figure B-4. Using Read/Write Strobes to Implement Address Decodes

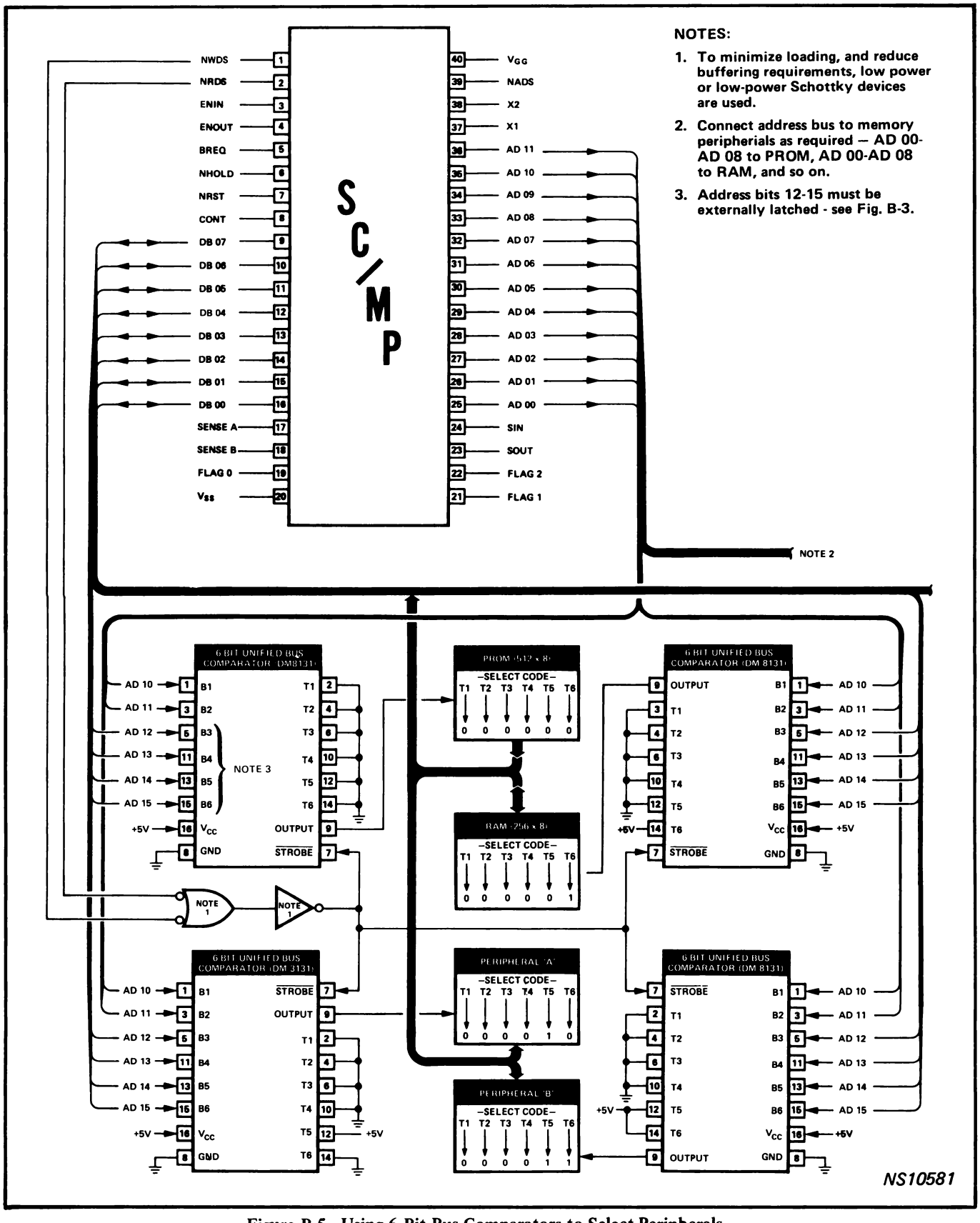


Figure B-5. Using 6-Bit Bus Comparators to Select Peripherals

APPENDIX C

SC/MP INTERRUPT SYSTEM

SINGLE-LEVEL INTERRUPTS

The interrupt system of SC/MP is supervised as shown in figure C-1. Before an instruction is fetched, bit 3 of the Status Register is tested. If the bit is a logic '0' (interrupt enable flag *not* set) and the CONTinue input is high (a logic '1'), the Program Counter is incremented and the next instruction is fetched and executed. If bit 3 of the Status Register *is* set to a logic '1' and the Sense A input is high, the interrupt is serviced; the interrupt enable flag (bit 3 of the Status Register) is reset and the contents of the Program Counter are exchanged with the contents of Pointer Register 3 – the pointer contains the address of the subroutine that services the interrupt.

In summary, Pointer Register 3 must be initialized with the address of the interrupt service routine, and then to arm the interrupt system, the interrupt enable flag (bit 3 of the Status Register) must be set to a logic '1'. For example, to enable the interrupt system at some point in the main program with the interrupt service routine residing at location X'1020, the following series of instructions could be implemented.

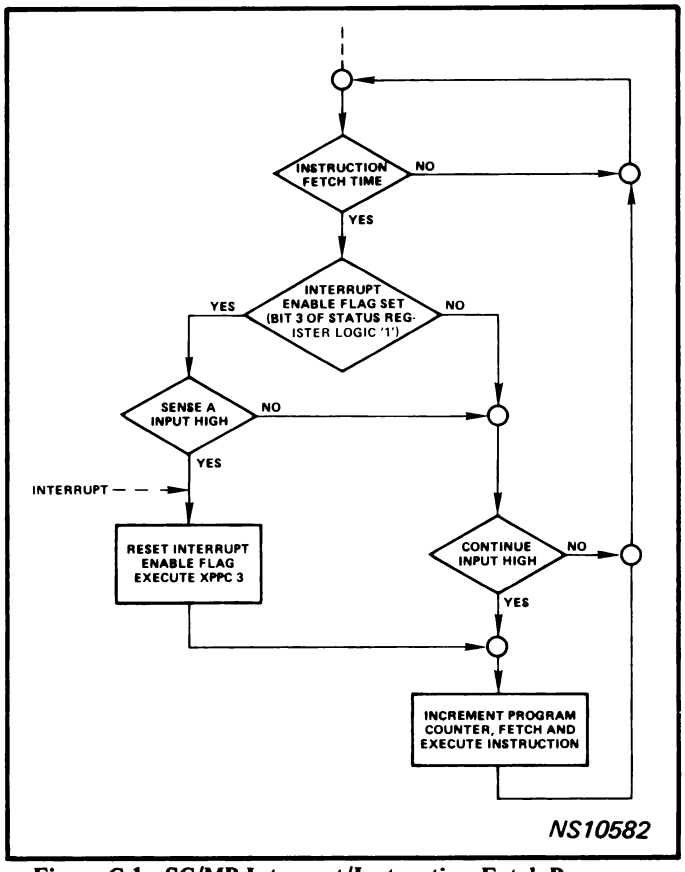


Figure C-1. SC/MP Interrupt/Instruction Fetch Process

```
Main Program
    •
    •
    •
    LDI     X'20    ; LOAD ACCUMULATOR WITH LOW-
                    ; ORDER BYTE (X'20) OF INTERRUPT
                    ; SERVICE ROUTINE ADDRESS

    XPAL    P3      ; PUT LOW-ORDER BYTE IN P3L
    LDI     X'10    ; LOAD ACCUMULATOR WITH HIGH-
                    ; ORDER BYTE (X'10) OF INTERRUPT
                    ; SERVICE ROUTINE ADDRESS

    XPAH    P3      ; PUT HIGH-ORDER BYTE IN P3H

    IEN             ; ENABLE INTERRUPTS (SET BIT 3
                    ; OF STATUS REGISTER TO LOGIC '1')

    •
    •
    •
Continue Main
Program
```

MULTI-LEVEL INTERRUPTS

A system with more than one interrupt level can be easily implemented by interfacing a priority encoder to SC/MP. In such systems, the interrupt service routine for each interrupting device should be carefully written to avoid complicated software conflicts; this is particularly true in saving the "current" status of the machine when the interrupt occurs. The "saving-of-status" can be quite complicated, especially in a nested prioritized interrupt system; in this case, the higher priority interrupt level can interrupt the service routine of the lower priority interrupt. Since "nested" interrupt routines are a function of the user's software, implementation techniques are not described in this document.

Figure C-2 shows one method of interfacing a priority encoder to SC/MP. Each of the eight inputs (INT 0 through INT 7) should come from a latched output which is reset by SC/MP during execution of the interrupt service routine.

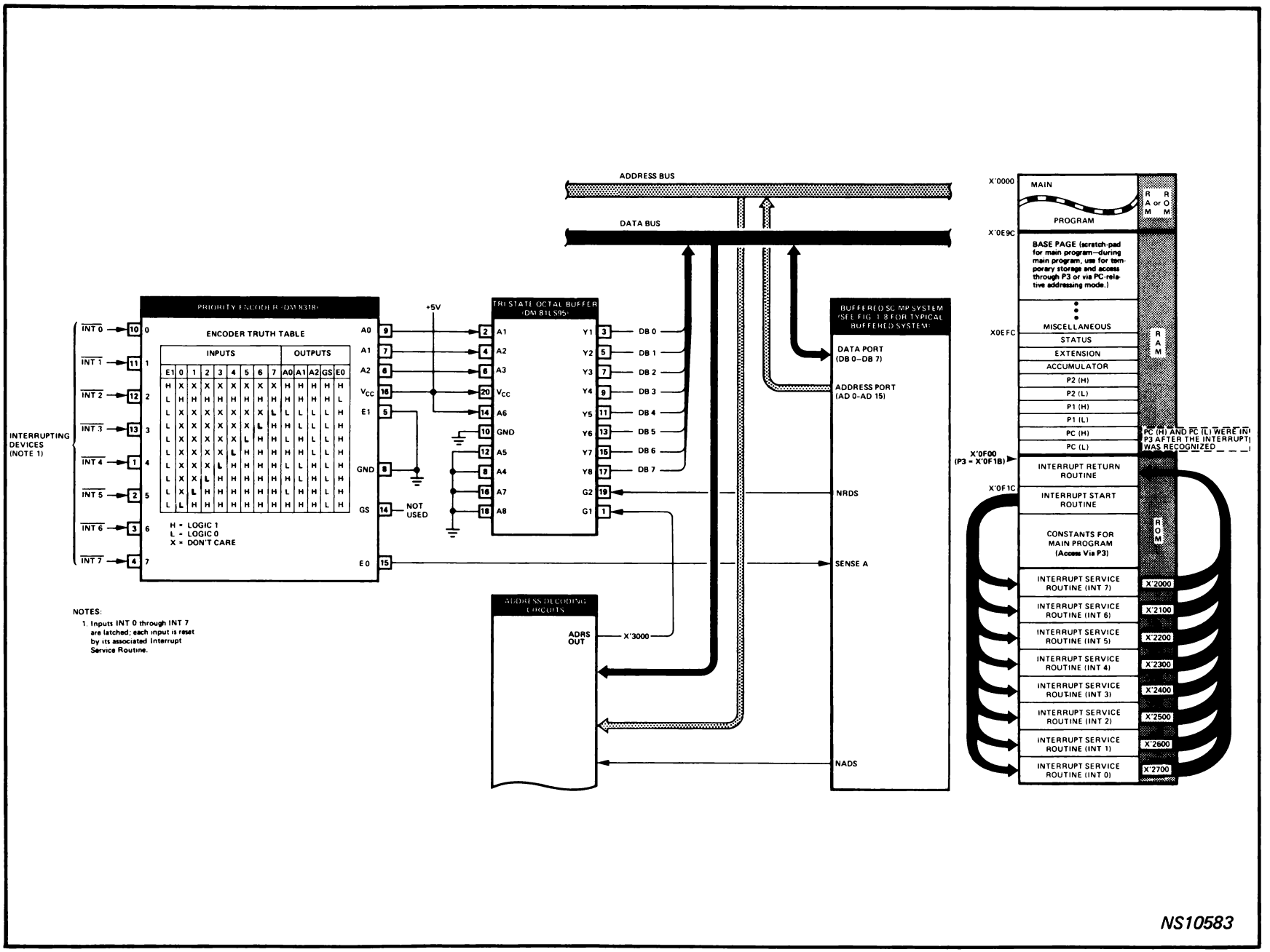


Figure C-2. Using SC/MP and a Priority Encoder to Implement a Multilevel Interrupt System

The encoder "ORs" the interrupt inputs and, if any one is "active-low," the Sense A (interrupt mode) line of SC/MP is raised to a logic '1' via the E0 output line (pin 15). If two or more inputs request interrupt service at the same time, they are assigned the service on a prioritized basis – INT 7 being the highest priority interrupt and INT 0 being the lowest. The A0, A1, and A2 outputs of the encoder are a 3-bit binary code that corresponds to the highest priority interrupt; this code forms the least significant bits of the interrupt data word (0010 0XXX) and is input to SC/MP via the TRI-STATE buffer. The output of the buffer provides the upper 8-bits of the 16-bit address required for each interrupt service routine. As shown in figure C-2, the highest priority interrupt (INT 7) is arbitrarily assigned an address of X'2000; the next highest priority (INT 6) – an address of X'2100; (INT 5) – an address of X'2200; and so on for the remaining interrupt levels. The output data word for each interrupt level is shown below.

Interrupt Level	Interrupt Buffer Output Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	2	1	0
Interrupt #7	0	0	1	0	0	0	0	0
#6	0	0	1	0	0	0	0	1
#5	0	0	1	0	0	0	1	0
#4	0	0	1	0	0	0	1	1
#3	0	0	1	0	0	1	0	0
#2	0	0	1	0	0	1	0	1
#1	0	0	1	0	0	1	1	0
#0	0	0	1	0	0	1	1	1

Memory allocations for the eight interrupt service routines (INT 0–INT 7) are also shown in figure C-2. These allocations include the *interrupt start routine*, the *interrupt return routine*, and *temporary storage for status information*. The addresses are chosen such that these locations fall within the same page boundary – this arrangement permits the use of PC-relative addressing for the start/return routines.

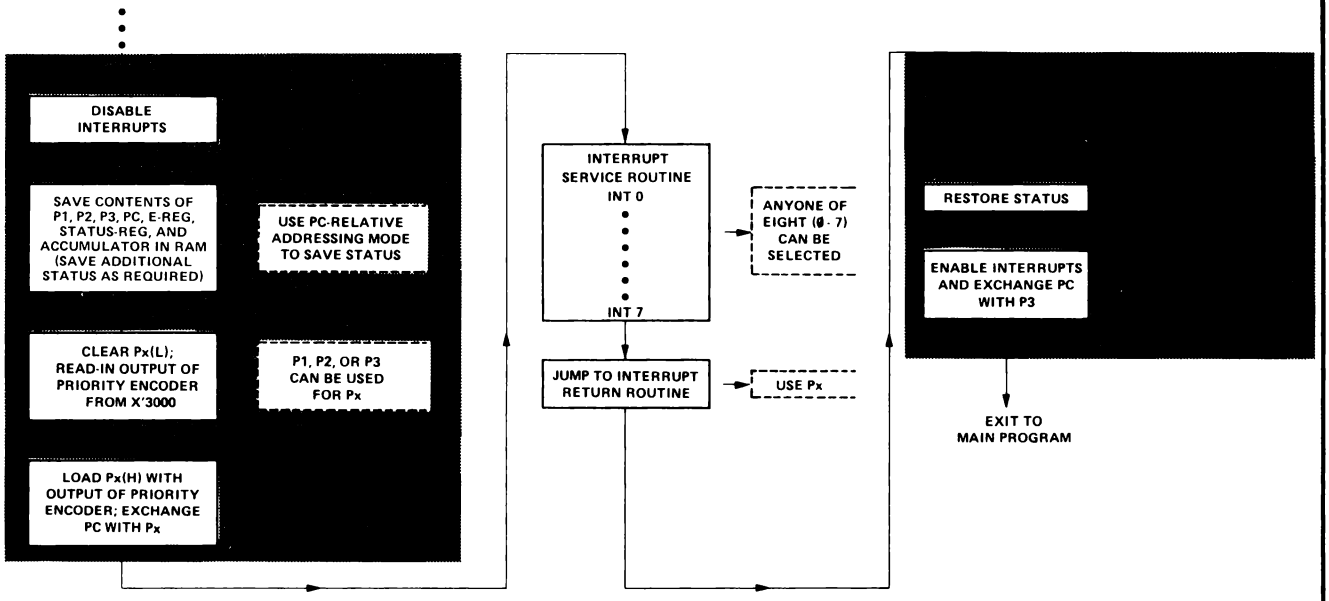
The *interrupt start routine* is written to save the status of the machine (Accumulator, Program Counter, Pointer Registers, and so on) and to direct the program to the interrupt service routine of the highest priority interrupt. To transfer control to the service routine, the program clears the lower half of a pointer register – in the example shown, Pointer Register P2. The program then reads in the address data word (0010 0XXX) of the highest priority interrupt; this information is transferred to the upper half of a pointer register, which provides a 16-bit address of the required interrupt service routine. At this point, the Program Counter and Pointer Register are exchanged and the interrupt service routine is executed. As shown in figure C2, each interrupt must go through the start routine to execute the foregoing steps. *(Note: Once an interrupt is recognized, all pending interrupts–even those with higher priority – must wait for service.*

Interrupt service routines 0 through 7 contain a program to service the particular interrupting device. Typically, these programs perform input/output transfer of data, exercise electrical/mechanical control of peripherals, furnish timing synchronization, and/or other relevant functions. The flowchart and program listing (figure C-3) is a flexible and general-purpose scheme that represents one of many ways to implement a multilevel interrupt system using SC/MP. For example, the interrupt service routines can be started at locations other than X'2000 by simply rearranging the inputs to the TRI-STATE buffer in figure C-2. Also, the service routines need not be 256 words in length; the number of words can be altered by rearranging the buffer inputs and by loading the lower half of the pointer instead of the upper half.

Upon completion of the interrupt service routine, all interrupts must go through the *interrupt return routine*. This routine essentially restores the status of machine as it was before the interrupt was recognized. The interrupt enable flag (bit 3 of the Status Register) is set to service interrupts that are still pending. The interrupt start and return routines could be changed to save more or less status than shown. Also, some of the status words can be changed; these changes depend largely on what is done during the interrupt service routine.

In the preceding example of multiple interrupts, it should be noted that P3 is used in a unique manner. Pointer Register P3 is used as the interrupt service pointer and is loaded with X'0F1B. Referring to the memory map the ROM/RAM boundary was chosen to be X'0F00 which is close to the pointer register value. When the Pointer Register is exchanged with the Program Counter due to an interrupt, the status is saved in RAM by use of PC-relative addressing. At the end of the service routine, the status is restored by use of PC-relative addressing.

Due to the choice of the value for the Pointer Register when the main program is being executed, P3 may be used as a pointer for temporary (scratch pad) storage by the main program by the use of indexed addressing. The area available for scratch pad in this example is the area between the end of the save area (X'0EF6) and -127 from the value in the Pointer Register (X'0F1B – 7F) or X'0E9C. Pointer Register 3 may also be used to access constants stored in ROM at the end of the interrupt start routine via the indexed addressing mode. The area available for constant storage is X'0F40 to X'0F9A (X'0F1B + 7F).




```
 1                          TITLE  INTRPT,   VECTORED INTERUPT FOR SC/MP'
 2
 3        0001  P1        =       1
 4        0002  P2        =       2
 5        0003  P3        =       3
 6
 7  0000  08              NOP
 8  0001  C41B  MAIN:     LDI     L(INTRPT)-1     ;SET UP INTERRUPT POINTER
 9  0003  33              XPAL    P3
10  0004  C40F            LDI     H(INTRPT)
11  0006  37              XPAH    P3
12  0007  05              IEN                     ;ENABLE INTERRUPT
13
14                ;USER MAIN PROGRAM STARTS HERE
15
16
17        0EF6             =0EF6
18
19        0EF7  STATUS:    =.+1
20        0EF8  EXT:       =.+1
21        0EF9  ACU:       =.+1
22        0EFA  P2H:       =.+1
23        0EFB  P2L:       .=.+1
24        0EFC  P1H:       .=.+1
25        0EFD  P1L:       .=.+1
26        0EFE  PCH:       .=.+1
27        0EFF  PCL:       .=.+1
28
29                         .PAGE    'INTERRUPT RETURN'
30
31        0F00             =0F00
32
```

Figure C–3. Flowchart and Program Listing for Multilevel Interrupt System

```
33                      ;INTERRUPT RETURN ROUTINE.  RESTORES SYSTEM STATUS
34                      ;AND RETURNS TO MAIN PROGRAM.
35
36 0F00 C0F6  INTRTN:   LD      EXT             ;RESTORE EXT REG
37 0F02 01              XAE
38 0F03 C0F0            LD      P1L             ;RESTORE P1
39 0F05 31              XPAL    P1
40 0F06 C0F4            LD      P1H
41 0F08 35              XPAH    P1
42 0F09 C0F0            LD      P2L             ;RESTORE P2
43 0F0B 32              XPAL    P2
44 0F0C C0EC            LD      P2H
45 0F0E 36              XPAH    P2
46 0F0F C0EE            LD      PCL             ;RESTORE RETURN ADDR.
47 0F11 33              XPAL    P3
48 0F12 C0EA            LD      PCH
49 0F14 37              XPAH    P3
50 0F15 C0E0            LD      STATUS          ;RESTORE STATUS REG.
51 0F17 07              CAS
52 0F18 C0DF            LD      ACU             ;RESTORE ACU
53 0F1A 05              IEN
54 0F1B 3F              XPPC    P3
55
56
                         PAGE   'INTERRUPT ROUTINE'
57
58                      ;INTERRUPT ROUTINE.  READS PRIORITY ENCODER TO DETERMINE
59                      ;INTERRUPT NEEDING SERVICE AND BRANCHES THERE.
60
61
62 0F1C C8DB  INTRPT:   ST      ACU             ;SAVE ACU
63 0F1E 06              CSA
64 0F1F C8D6            ST      STATUS          ;SAVE STATUS REG.
65 0F21 33              XPAL    P3              ;SAVE RETURN ADDR.
66 0F22 C8DB            ST      PCL
67 0F24 37              XPAH    P3
68 0F25 C8D7            ST      PCH
69 0F27 31              XPAL    P1              ;SAVE P1
70 0F28 C8D3            ST      P1L
71 0F2A 35              XPAH    P1
72 0F2B C8CF            ST      P1H
73 0F2D 32              XPAL    P2              ;SAVE P2
74 0F2E C8CB            ST      P2L
75 0F30 36              XPAH    P2
76 0F31 C8C7            ST      P2H
77 0F33 01              XAE
78 0F34 C8C2            ST      EXT             ;SAVE EXT REG.
79 0F36 C400  RDSTAT:   LDI     0               ;CLEAR P2
80 0F38 32              XPAL    P2
81 0F39 C430            LDI     030             ;SET ADDR OF ENCODER
82 0F3B 36              XPAH    P2
83 0F3C C200            LD      (P2)            ;READ PRIORITY ENCODER
84 0F3E 36              XPAH    P2              ;SET VECTOR IN P2
```

Figure C–3 (Continued)

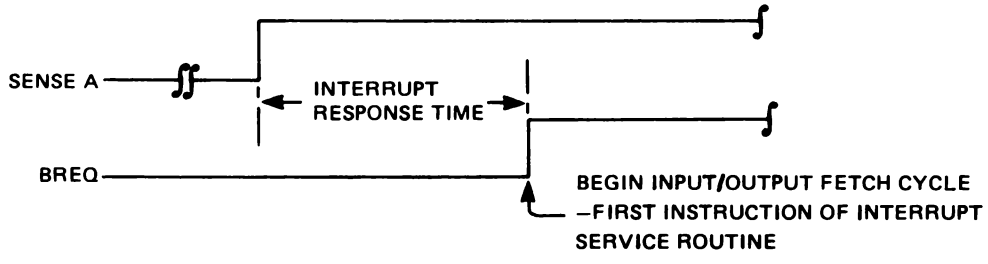
```
85 0F3F 3E              XPPC    P2          ;GO TO SERVICE ROUTINE
86
87
88
89          ;INTERRUPT SERVICE ROUTINE RETURN. SERVICE ROUTINE EXECUTES
90          ;"XPPC P2" INSTRUCTION.
91
92 0F40 90BE   SRETRN: JMP     INTRTN     ;RETURN FROM SERVICE, GO TO
93                                         ;INTERUPT RETURN
94
95        0000            END

        ACU     0EF8        EXT     0EF7        INTRPT  0F1C
        INTRTN  0F00        MAIN    0001 *      P1      0001
        P1H     0EFB        P1L     0EFC        P2      0002
        P2H     0EF9        P2L     0EFA        P3      0003
        PCH     0EFD        PCL     0EFE        RDSTAT  0F36 *
        SRETRN  0F40 *      STATUS  0EF6


        NO ERROR LINES
        SOURCE CHECKSUM=C25E
```

NS10584

Figure C–3 (Concluded)

INTERRUPT RESPONSE TIME

For both single-level and multilevel interrupts, the interrupt response time (IRT) is an important parameter. As shown in figure C-1, an interrupt can occur with the CONT input high and SC/MP running or with the CONT input low and SC/MP in the halt mode. The maximum interrupt response time for an operation when SC/MP is executing an instruction of a main program is equal to the *Maximum (worst case) Execution Time + Overhead*. The execution time for instruction DAD requires 23 machine cycles ($23 \times 2T_X$), where T_X is a clock cycle that is equal to 1 microsecond for a 1-megahertz clock frequency. The overhead based on the microcodes equals $14T_X + 200$ nanoseconds. The total maximum interrupt response time equals $46T_X + (14T_X + 200$ nanoseconds); this time does not include the "delay" instruction because of its variable nature and does not include the "hold" operation. In general, the interrupt response time depends upon the instruction that is being executed. For a more precise interrupt response time, the system configuration can put SC/MP in the HALT mode by driving the CONTinue line low and keeping the interrupts enabled. Now, when an interrupt occurs, the response time is guaranteed to be $12T_X \leqslant T_{ir} \leqslant (14T_X + 200$ nanoseconds).



APPENDIX D
MATH ROUTINES

The SC/MP instruction set (table 1-2) provides the following 'math' capabilities. The routines (tables D-1 through D-8) are allocated contiguously in memory; however, any one of the routines can be used discretely by simply altering the end-of-routine 'jump (JMP) address' to agree with a particular applications program. For programming techniques and other software information, the user is referred to the 'SC/MP Assembly Language Programming Manual—order number ISP-85/994Y.'

PREFACE MATERIAL

```
 1                       TITLE  MATH    SC/MP MATH ROUTINES'
 2                     12/29/75
 3                     SC/MP MATH ROUTINES
 4
 5                     ALL MATH ROUTINES WORK WITH THE STACK
 6
 7               ;     OPERANDS ARE 'PULLED' FROM THE STACK, AND THE
 8               ;     RESULT IS THEN 'PUSHED' ONTO THE STACK.
 9
10                     IN ALL EXAMPLES, OP1 REFERS TO THE TOP BYTE OF
11                     THE STACK, RELATIVE TO THE ORIGINAL POINTER;
12                     OP2 IS THE NEXT BYTE, ETC.
13
14
15                     STACK EXAMPLE FOR DOUBLE-ADD:
16               ;
17               ;     DADD: (OP3,OP4) = (OP1,OP2) + (OP3,OP4)
18                     STACK IS FROM 0000 TO 001F IN THIS EXAMPLE:
19
20                       001B  OP1 (HIGH1)      <-- STACK POINTER ON ENTRY
21                       001C  OP2 (LOW1)
22                       001D  OP3 (HIGH2)      <-- STACK POINTER ON EXIT
23                       001E  OP4 (LOW2)
24                       001F  <BOTTOM OF STACK>
25               ;
26               ;
27               ;     REGISTERS A AND E (AND P1 IF USED) ARE NOT SAVED.
28                     REGISTER P2 IS USED AS THE STACK POINTER.
29                     REGISTER P3 IS THE SUBROUTINE CALLING REGISTER.
30
31                     ALL OF THESE ROUTINES MAY BE REPEATED WITHOUT
32                     RELOADING P3 AFTER THE FIRST CALL.
33
34
```

TABLE D-1. DOUBLE ADD (DADD), DOUBLE NEGATE (DNEG), AND DOUBLE SUBTRACT (DSUB)

```
                        PAGE    'DADD, DNEG, DSUB'
35        1000          =       01000
36               ;
37               ;     DADD: DOUBLE ADD
38               ;           (OP3,OP4) = (OP1,OP2) + (OP3,OP4)
39               ;
40 1000 08     DADD:   NOP
41 1001 02             CCL
42 1002 C201           LD      1(2)            ; ADD LOW ORDER BYTE
43 1004 F203           ADD     3(2)            ;     (CARRY MAY BE SET)
44 1006 CA03           ST      3(2)            ;     SAVE RESULT
45 1008 C200           LD      0(2)            ; ADD HIGH ORDER BYTES + CARRY
46 100A F202           ADD     2(2)
47 100C CA02           ST      2(2)
48 100E C602           LD      02(2)           ; FIX STACK POINTER
49 1010 3F             XPPC    3
```

```
50 1011 90ED            JMP     DADD                JUMP FOR RECALL
51
52
```

```
53                      DNEG: DOUBLE NEGATE (2'S COMPLEMENT)
54              ;               (OP1,OP2) = - (OP1,OP2)
55              ;
56 1013 08      DNEG:   NOP
57 1014 03              SCL                         SET CARRY IN
58 1015 C400            LDI     0
59 1017 FA01            CAD     1(2)                NEGATE LOW BYTE & SET CARRY
60 1019 CA01            ST      1(2)
61 101B C400            LDI     0                   COMPLEMENT HIGH BYTE AND
62 101D FA00            CAD     0(2)                   ADD CARRY IN
63 101F CA00            ST      0(2)
64 1021 3F              XPPC    3
65 1022 90EF            JMP     DNEG
66              ;
67              ;
```

```
68              ;       DSUB: DOUBLE SUBTRACT
69              ;               (OP3,OP4) = (OP3,OP4) - (OP1,OP2)
70              ;
71 1024 08      DSUB:   NOP
72 1025 03              SCL                     ; SET CARRY IN (BORROW)
73 1026 C203            LD      3(2)            ; ADD LOW BYTES -
74 1028 FA01            CAD     1(2)            ;     OP4 + (-OP2)
75 102A CA03            ST      3(2)
76 102C C202            LD      2(2)              DO HIGH BYTES
77 102E FA00            CAD     0(2)
78 1030 CA02            ST      2(2)
79 1032 C602            LD      @2(2)           ; FIX POINTER
80 1034 3F              XPPC    3
81 1035 90ED            JMP     DSUB
82
```

TABLE D-2. UNSIGNED MULTIPLY

```
                         PAGE   'UNSIGNED MULTIPLY'
83                       LOCAL
84              ;
85              ;
86              ;       MPY: UNSIGNED MULTIPLY
87              ;               (OP1,OP2) = (OP1) * (OP2)
88              ;
89              ;       ONE BYTE OF STACK MEMORY IS USED FOR
90              ;       TEMPORARY STORAGE: -1(2)= BIT COUNT
91              ;
92              ;       EXECUTION TIME: ABOUT 2.5 MILLISEC.
93              ;       MAX TIME: ABOUT 3 MILLISEC.
94
95 1037 08      MPY:    NOP
96 1038 C200            LD      0(2)            ; GET MULTIPLIER
```

```
 97 103A 01             XAE                     ; SAVE IN E-REG
 98 103B C400           LDI     0
 99 103D CA00           ST      0(2)            ; ZERO OP1
100 103F C408           LDI     8
101 1041 CAFF           ST      -1(2)            ; SAVE COUNTER
102 1043 40     $LOOP:  LDE                     ; LOAD MULTIPLIER
103 1044 D401           ANI     1               ; TEST LEAST BIT
104 1046 9815           JZ      NO              ; BIT IS NOT ONE -
105 1048 C200           LD       0(2)           ; ADD MULTIPLICAND (8 BITS)
106 104A F201           ADD     1(2)            ;    TO 16-BIT RESULT
107             NOADD:
108 104C 02             CCL                     ;CLEAR LINC
109 104D 1F             RRL                     ;SHIFT
110 104E CA00           ST      0(2)
111 1050 40             LDE
112 1051 1F             RRL                     ;ROTATE
113 1052 01             XAE
114 1053 BAFF           DLD     -1(2)           ;DECREMENT COUNTER
115 1055 9CEC           JNZ     $LOOP
116 1057 40             LDE
117 1058 CA01           ST      1(2)
118 105A 3F             XPPC    3               ;RETURN
119 105B 90DA           JMP     MPY
120 105D C200   NO:     LD      0(2)
121 105F 90EB           JMP     NOADD
122
123
```

TABLE D-3. SIGNED MULTIPLY

```
                        .PAGE   'SIGNED MULTIPLY'
124             ;
125             ;       SMPY: SIGNED MULTIPLY
126             ;               (OP1,OP2) = (OP1) * (OP2)
127             ;               WHERE OP1 AND OP2 ARE SIGNED NUMBERS
128             ;               IN THE RANGE -128 TO 127
129             ;
130 1061 08     SMPY:   NOP
131 1062 C410           LDI     H(MPY)          ; SET P3 TO CALL MPY
132 1064 37             XPAH    3               ; WHILE SAVING ORIGINAL P3 IN
133 1065 35             XPAH    1
134 1066 C437           LDI     L(MPY)
135 1068 33             XPAL    3
136 1069 31             XPAL    1
137 106A 02             CCL                     ; COMPARE SIGNS
138 106B C200           LD      0(2)
139 106D E201           XOR     1(2)
140 106F 9404           JP      $SAME
141 1071 C4FF           LDI     -1
142 1073 9002           JMP     $MPY
143 1075 C400   $SAME:  LDI     0
144 1077 CAFE   $MPY:   ST      -2(2)
145 1079 C200           LD      0(2)            ; CHECK SIGN OF EACH
146 107B 9407           JP      $MPY2
```

```
147 107D 03             SCL
148 107E C400           LDI     0
149 1080 FA00           CAD     0(2)            ; NEGATE 1ST OPERAND
150 1082 CA00           ST      0(2)
151 1084 C201   $MPY2:  LD      1(2)
152 1086 9407           JP      $MPY3
153 1088 03             SCL
154 1089 C400           LDI     0
155 108B FA01           CAD     1(2)
156 108D CA01           ST      1(2)
157 108F 3F     $MPY3:  XPPC    3               ; DO UNSIGNED MULTIPLY
158 1090 C2FE           LD      -2(2)           ; CHECK SIGN FLAG
159 1092 940D           JP      $MPY4
160 1094 03             SCL
161 1095 C400           LDI     0
162 1097 FA01           CAD     1(2)            ; NEGATE RESULT
163 1099 CA01           ST      1(2)
164 109B C400           LDI     0
165 109D FA00           CAD     0(2)
166 109F CA00           ST      0(2)
167 10A1 35     $MPY4:  XPAH    1               ; GET RETURN ADDR FROM P1
168 10A2 37             XPAH    3
169 10A3 31             XPAL    1
170 10A4 33             XPAL    3
171 10A5 3F             XPPC    3               ; RETURN
172 10A6 90B9           JMP     SMPY
173
```

TABLE D-4. UNSIGNED DIVIDE

```
                        PAGE    'UNSIGNED DIVIDE'
174                     .LOCAL
175             ;
176             ;       DIV: UNSIGNED DIVIDE
177             ;
178             ;       DIVIDE A 16-BIT UNSIGNED NUMBER BY AN 8-BIT
179             ;       UNSIGNED NUMBER.  RESULT IS A 16-BIT QUOTIENT
180             ;       AND AN 8-BIT REMAINDER.
181             ;
182             ;       EXECUTION TIME: ABOUT 5 TO 20 MILLISEC.
183             ;
184             ;       CALL:   XPPC  3
185             ;               <ERROR RETURN>
186             ;               <NORMAL RETURN>
187             ;
188             ;       STACK USAGE:
189             ;               REL  ENTRY        USE          RETURN
190             ;               ---  ---------- ---------- ----------
191             ;                -5              TEMP
192             ;                -4              COUNT
193             ;                -3              REM(H)
194             ;                -2              REM(L)
195             ;                -1              DIVISOR(H)
196             ;          (P2)->  0  DIVISOR    DIVISOR(L)  REMAINDER
```

```
197                ;                    1  DIVIDEND(H) QUOTIENT(H) QUOTIENT(H)
198                ;                    2  DIVIDEND(L) QUOTIENT(L) QUOTIENT(L)
199                ;
200                ;
201 10A8 08     DIV:     NOP
202 10A9 C200              LD       0(2)                 CHECK FOR ZERO DIVISOR
203 10AB 9C03              JNZ      $D1
204 10AD 3F                XPPC     3                  ; ERROR - RETURN
205 10AE 90F8              JMP      DIV
206 10B0 C409   $D1:       LDI      9                    COUNT = 9
207 10B2 CAFC              ST       -4(2)
208 10B4 C200              LD       0(2)
209 10B6 01                XAE
210 10B7 40     $SET:      LDE                           ; NORMALIZE THE DIVISOR
211 10B8 9402              JP        +4
212 10BA 9007              JMP      $SETUP
213 10BC 02                CCL                           ; SHIFT LEFT 1 BIT
214 10BD 70                ADE
215 10BE 01                XAE
216 10BF AAFC              ILD      -4(2)                ; COUNT = COUNT + 1
217 10C1 90F4              JMP      $SET
218                ;
219 10C3 40     $SETUP:    LDE
220 10C4 CAFF              ST       -1(2)                ; SAVE DIVISOR
221 10C6 C201              LD       1(2)
222 10C8 CAFD              ST       -3(2)                ; COPY DIVIDEND TO
223 10CA C202              LD       2(2)                 ;    INITIAL REMAINDER
224 10CC CAFE              ST       -2(2)
225 10CE C400              LDI      0
226 10D0 CA00              ST       0(2)                 ; INITIALIZE LOW BYTE OF DIVIS
227 10D2 CA01              ST       1(2)                 ;    AND RESULT TO ZERO
228 10D4 CA02              ST       2(2)
229                ;
230 10D6 03     $LOOP:     SCL
231 10D7 C2FE              LD       -2(2)                ; SUBTRACT:
232 10D9 FA00              CAD      0(2)                 ;    REMAINDER-DIVISOR
233 10DB 01                XAE
234 10DC C2FD              LD       -3(2)
235 10DE FAFF              CAD      -1(2)
236 10E0 CAFB              ST       -5(2)                ; SAVE TEMPORARILY
237 10E2 06                CSA                           ; CHECK CARRY:
238 10E3 E480              XRI      080
239 10E5 941E              JP       $DVGR                ; JUMP IF RESULT >= 0
240 10E7 BAFC              DLD      -4(2)                ; COUNT = COUNT - 1
241 10E9 9829              JZ       $DONE                ; CHECK IF DONE
242 10EB 02                CCL                           ;    NO
243 10EC C202              LD       2(2)                 ; DOUBLE LEFT SHIFT:
244 10EE F202              ADD      2(2)                 ;    ADD QUOTIENT TO ITSELF
245 10F0 CA02              ST       2(2)
246 10F2 C201              LD       1(2)
247 10F4 F201              ADD      1(2)
248 10F6 CA01              ST       1(2)
249 10F8 02                CCL
250 10F9 C2FF              LD       -1(2)                ; SHIFT DIVISOR RIGHT 1 BIT
251 10FB 1F                RRL
```

```
252 10FC CAFF            ST       -1(2)
253 10FE C200            LD       0(2)
254 1100 1F              RRL
255 1101 CA00            ST       0(2)
256 1103 90D1            JMP      $LOOP
257                ;
258 1105 C2FB  $DVGR:    LD       -5(2)              SAVE NEW REMAINDER
259 1107 CAFD            ST       -3(2)                 FOLLOWING SUBTRACTION
260 1109 01              XAE
261 110A CAFE            ST       -2(2)
262 110C AA02            ILD      2(2)            ; INCREMENT QUOTIENT
263 110E 9CC6            JNZ      $LOOP
264 1110 AA01            ILD      1(2)
265 1112 90C2            JMP      $LOOP
266                ;
267 1114 C2FE  $DONE:    LD       -2(2)           ; DONE: COPY FINAL REMAINDER
268 1116 CA00            ST       0(2)
269 1118 C702            LD       @2(3)           ; INCREMENT P3 FOR NORMAL RETU
270 111A 3F              XPPC     3
271 111B 908B            JMP      DIV
272
273
```

TABLE D-5. BCD MULTIPLY

```
                          PAGE    'BCD MULTIPLY'
274                       .LOCAL
275                ;
276                ;      BCDMPY: MULTIPLY TWO 6-DIGIT BCD NUMBERS
277                ;              RESULT IS 12 DIGITS
278                ;
279                ;      EXECUTION TIME: 6 TO 27 MILLISEC.   (17 TYPICAL)
280                ;
281                ;      STACK USAGE:
282                ;              REL  ENTRY        USE          RETURN
283                ;              ---  ------------ ------------ ------------
284                ;              -10               TEMP
285                ;               -9               PRODUCT(1)
286                ;               -8               PRODUCT(2)
287                ;               -7               PRODUCT(3)
288                ;               -6               PRODUCT(4)
289                ;               -5               PRODUCT(5)
290                ;               -4               PRODUCT(6)
291                ;               -3               COUNT
292                ;               -2               M1(1)
293                ;               -1               M1(2)
294                ;       (P2)->   0  M1(SIGN)     M1(3)
295                ;                1  M1(1)        M1(4)        PRODUCT(SIGN)
296                ;                2  M1(2)        M1(5)        PRODUCT(1)
297                ;                3  M1(3)        M1(6)        PRODUCT(2)
298                ;                4  M2(SIGN)     PROD(SIGN)   PRODUCT(3)
299                ;                5  M2(1)        M2(1)        PRODUCT(4)
300                ;                6  M2(2)        M2(2)        PRODUCT(5)
301                ;                7  M2(3)        M2(3)        PRODUCT(6)
```

```
302
303                     ;
304         FFFE    M1      =       -2              ; MULTIPLICAND (6 WORDS)
305         0005    M2      =       5               ; MULTIPLIER   (3 WORDS)
306         FFF7    PR      =       -9              ; PRODUCT (6 WORDS)
307         FFFD    COUNT   =       -3              ; DIGIT COUNT (1 WORD)
308         FFF6    TEMP    =       -10             ; SHIFT COUNT OR MULTIPLIER DI
309         FFF6    M3      =       -10             ; MULTIPLIER DIGIT
310
311                     ;
312 111D 08      BCDMPY: NOP
313 111E C200          LD      0(2)            ; CALCULATE SIGN OF PRODUCT
314 1120 02            CCL
315 1121 F204          ADD     4(2)
316 1123 D401          ANI     1
317 1125 CA04          ST      4(2)            ;    AND SAVE IT FOR LATER
318 1127 C400          LDI     0               ; INITIALIZE PRODUCT
319 1129 CAF7          ST      PR(2)           ;    AND MULTIPLICAND EXTENSIO
320 112B CAF8          ST      PR+1(2)
321 112D CAF9          ST      PR+2(2)
322 112F CAFA          ST      PR+3(2)
323 1131 CAFB          ST      PR+4(2)
324 1133 CAFC          ST      PR+5(2)
325 1135 CAFE          ST      M1(2)
326 1137 CAFF          ST      M1+1(2)
327 1139 CA00          ST      M1+2(2)
328 113B C406          LDI     6               ; DIGIT COUNT = 6
329 113D CAFD          ST      COUNT(2)
330 113F C207  $LOOP:  LD      M2+2(2)         ; LOAD MULTIPLIER
331 1141 D40F          ANI     0F              ;    USE LOWEST BCD DIGIT
332 1143 982B          JZ      NEXTD           ; IF ZERO THEN GO TO NEXT DIGI
333 1145 CAF6          ST      M3(2)
334 1147 02    $L1:    CCL                     ; ADD MULTIPLICAND TO PRODUCT
335 1148 C203          LD      M1+5(2)
336 114A EAFC          DAD     PR+5(2)
337 114C CAFC          ST      PR+5(2)
338 114E C202          LD      M1+4(2)
339 1150 EAFB          DAD     PR+4(2)
340 1152 CAFB          ST      PR+4(2)
341 1154 C201          LD      M1+3(2)
342 1156 EAFA          DAD     PR+3(2)
343 1158 CAFA          ST      PR+3(2)
344 115A C200          LD      M1+2(2)
345 115C EAF9          DAD     PR+2(2)
346 115E CAF9          ST      PR+2(2)
347 1160 C2FF          LD      M1+1(2)
348 1162 EAF8          DAD     PR+1(2)
349 1164 CAF8          ST      PR+1(2)
350 1166 C2FE          LD      M1(2)
351 1168 EAF7          DAD     PR(2)
352 116A CAF7          ST      PR(2)
353 116C BAF6          DLD     M3(2)           ; DECREMENT MULTIPLIER
354 116E 9CD7          JNZ     $L1             ;    AND ADD AGAIN IF NOT ZERO
355 1170 BAFD  NEXTD:  DLD     COUNT(2)        ; DECREMENT DIGIT COUNT
356 1172 9849          JZ      $OUT            ;    QUIT IF DONE
```

```
357 1174 C404           LDI     4               ; SHIFT MULTIPLICAND LEFT
358 1176 CAF6           ST      TEMP(2)         ;    4 BITS (1 DIGIT)
359 1178 02     $L2:    CCL
360 1179 C203           LD      M1+5(2)
361 117B F203           ADD     M1+5(2)
362 117D CA03           ST      M1+5(2)
363 117F C202           LD      M1+4(2)
364 1181 F202           ADD     M1+4(2)
365 1183 CA02           ST      M1+4(2)
366 1185 C201           LD      M1+3(2)
367 1187 F201           ADD     M1+3(2)
368 1189 CA01           ST      M1+3(2)
369 118B C200           LD      M1+2(2)
370 118D F200           ADD     M1+2(2)
371 118F CA00           ST      M1+2(2)
372 1191 C2FF           LD      M1+1(2)
373 1193 F2FF           ADD     M1+1(2)
374 1195 CAFF           ST      M1+1(2)
375 1197 C2FE           LD      M1(2)
376 1199 F2FE           ADD     M1(2)
377 119B CAFE           ST      M1(2)
378 119D BAF6           DLD     TEMP(2)
379 119F 9CD7           JNZ     $L2
380 11A1 C2FD           LD      COUNT(2)        ; GET NEXT MULTIPLIER DIGIT
381 11A3 D401           ANI     1               ; IF COUNT IS EVEN THEN USE NE
382 11A5 980A           JZ      $L3
383 11A7 C207           LD      M2+2(2)         ; OTHERWISE SHIFT LOW WORD RIG
384 11A9 1C             SR
385 11AA 1C             SR
386 11AB 1C             SR
387 11AC 1C             SR
388 11AD CA07           ST      M2+2(2)
389 11AF 908E           JMP     $LOOP
390 11B1 C206   $L3:    LD      M2+1(2)
391 11B3 CA07           ST      M2+2(2)
392 11B5 C205           LD      M2(2)
393 11B7 CA06           ST      M2+1(2)
394 11B9 C207           LD      M2+2(2)
395 11BB 9082           JMP     $LOOP
396 11BD C204   $OUT:   LD      4(2)            ; COPY SIGN TO OUTPUT
397 11BF CA01           ST      1(2)
398 11C1 C2F7           LD      PR(2)
399 11C3 CA02           ST      2(2)
400 11C5 C2F8           LD      PR+1(2)
401 11C7 CA03           ST      3(2)
402 11C9 C2F9           LD      PR+2(2)
403 11CB CA04           ST      4(2)
404 11CD C2FA           LD      PR+3(2)
405 11CF CA05           ST      5(2)
406 11D1 C2FB           LD      PR+4(2)
407 11D3 CA06           ST      6(2)
408 11D5 C2FC           LD      PR+5(2)
409 11D7 CA07           ST      7(2)
410 11D9 3F             XPPC    3               ; RETURN
411 11DA C411           LDI     H(BCDMPY)       ; JUMP TO BCDMPY
```

```
412 11DC 35             XPAH    1
413 11DD C41D           LDI     L(BCDMPY)
414 11DF 31             XPAL    1
415 11E0 91FF           JMP     0(1)
416
417
```

TABLE D-6. BCD ADD/SUBTRACT

```
                        PAGE    'BCD ADD/SUBTRACT'
418                     LOCAL
419             ;
420             ;       BCD ADDITION AND SUBTRACTION
421             ;       EACH OPERAND IS 6 DIGIT MAGNITUDE PLUS SIGN
422             ;
423             ;       EXECUTION TIME: 630 TO 1110 MICROSEC.
424             ;
425             ;       BCDADD: OP2 = OP2 + OP1
426             ;       BCDSUB: OP2 = OP2 - OP1
427             ;
428             ;               OP1 IS NOT ALTERED
429             ;
430             ;       SIGN BYTE: 0=POSITIVE, 1=NEGATIVE
431             ;
432             ;       CALL:   XPPC  3
433             ;               <OVERFLOW RETURN>
434             ;               <NORMAL RETURN>
435             ;
436             ;
437             ;       STACK USAGE:
438             ;               REL  ENTRY        RETURN
439             ;               ---  ------------ ------------
440             ;                 0  OP1(SIGN)    OP1(SIGN)
441             ;                 1  OP1(1)       OP1(1)
442             ;                 2  OP1(2)       OP1(2)
443             ;                 3  OP1(3)       OP1(3)
444             ;                 4  OP2(SIGN)    RESULT(SIGN)
445             ;                 5  OP2(1)       RESULT(1)
446             ;                 6  OP2(2)       RESULT(2)
447             ;                 7  OP2(3)       RESULT(3)
448             ;
449      0001   $OP1    =        1               ; FIRST OPERAND
450      0005   $OP2    =        5               ; SECOND OPERAND
451             ;
452             BCDSUB:
453 11E2 C201          LD       $OP1(2)         ;CHECK FOR ZERO OP1.
454 11E4 9C08          JNZ      $BSUB           ;NOT ZERO.
455 11E6 C202          LD       $OP1+1(2)       ;CHECK FOR ZERO OP1.
456 11E8 9C04          JNZ      $BSUB           ;NOT ZERO.
457 11EA C203          LD       $OP1+2(2)       ;CHECK FOR ZERO OP1.
458 11EC 9827          JZ       $OUT            ;OP1 IS ZERO RETURN.
459 11EE C200  $BSUB:  LD       $OP1-1(2)       ; CHANGE SIGN OF SUBTRAHEND
460 11F0 E401          XRI      01              ;     THEN DO ADDITION
461 11F2 01            XAE
```

```
462 11F3 9003           JMP     $CHK
463 11F5 C200  BCDADD:  LD      $OP1-1(2)       ; COMPARE SIGNS
464 11F7 01             XAE
465 11F8 C204  $CHK:    LD      $OP2-1(2)
466 11FA 60             XRE
467 11FB 9C21           JNZ     $DIFF           ; DIFFERENT SIGNS - JUMP
468 11FD 02    SAME:    CCL                     ; SAME SIGNS -
469 11FE C203           LD      $OP1+2(2)       ;    ADD MAGNITUDES
470 1200 EA07           DAD     $OP2+2(2)
471 1202 CA07           ST      $OP2+2(2)
472 1204 C202           LD      $OP1+1(2)
473 1206 EA06           DAD     $OP2+1(2)
474 1208 CA06           ST      $OP2+1(2)
475 120A C201           LD      $OP1(2)
476 120C EA05           DAD     $OP2(2)
477 120E CA05           ST      $OP2(2)
478 1210 06             CSA                     ; IS THERE AN END CARRY?
479 1211 D480           ANI     080
480 1213 9C02           JNZ     $OVFL           ; YES - OVERFLOW
481 1215 C702  $OUT:    LD      @2(3)           ; INCREMENT P3 BY 2 FOR NORMAL
482 1217 3F    $OVFL:   XPPC    3               ; RETURN
483
484
485            OP2Z:
486 1218 C401           LDI     1               ;SET OP2 SIGN NEG.
487 121A CA04           ST      $OP2-1(2)
488 121C 90DF           JMP     SAME            ;NOW ADD OP1 TO OP2.
489
490
491
492
493
494            ;
495            $DIFF:                           ; IF OP1 IS NEGATIVE THEN SET
496 121E 40             LDE                         TO GET OP2-OP1 = -(OP1-OP
497 121F 9802           JZ      $2                  JUMP IF OP1(SIGN) IS ZERO
498 1221 C401           LDI     1
499 1223 CA04  $2:      ST      $OP2-1(2)       ; SIGN FLAG IS IN SIGN.
500 1225 C205           LD      $OP2(2)         ;CHECK IF OP2 IS ZERO.
501 1227 9C08           JNZ     OK              ;NO.
502 1229 C206           LD      $OP2+1(2)
503 122B 9C04           JNZ     OK
504 122D C207           LD      $OP2+2(2)
505 122F 98E7           JZ      OP2Z            ;OP2 IS ZERO.
506 1231 C605  OK:      LD      @$OP2(2)            SET P2 TO POINT TO OP2
507 1233 C412           LDI     H(BCDCMP)           COMPLEMENT OP2
508 1235 35             XPAH    1
509 1236 C47D           LDI     L(BCDCMP)
510 1238 31             XPAL    1
511 1239 02             CCL
512 123A 3D             XPPC    1
513 123B C6FB           LD      @-$OP2(2)           RESTORE P2
514 123D 02             CCL                         ADD MAGNITUDES
515 123E C203           LD      $OP1+2(2)
516 1240 EA07           DAD     $OP2+2(2)
```

```
517 1242 CA07           ST      $OP2+2(2)
518 1244 C202           LD      $OP1+1(2)
519 1246 EA06           DAD     $OP2+1(2)
520 1248 CA06           ST      $OP2+1(2)
521 124A C201           LD      $OP1(2)
522 124C EA05           DAD     $OP2(2)
523 124E CA05           ST      $OP2(2)
524 1250 06             CSA                     ; IS THERE AN END CARRY?
525 1251 D480           ANI     080
526 1253 9C10           JNZ     $PLUS             YES - SIGN IS POSITIVE
527 1255 C605           LD      @$OP2(2)          SET P2 TO RESULT
528 1257 C412           LDI     H(BCDCMP)       ; 10'S COMPLEMENT THE RESULT
529 1259 35             XPAH    1
530 125A C47D           LDI     L(BCDCMP)
531 125C 31             XPAL    1
532 125D 02             CCL
533 125E 3D             XPPC    1
534 125F C6FB           LD      @-$OP2(2)       ; RESTORE P2
535 1261 C401           LDI     1               ; SIGN IS NEGATIVE
536 1263 9002           JMP     $RTN2
537 1265 C400   $PLUS:  LDI     0               ; SIGN IS POSITIVE
538 1267 E204   $RTN2:  XOR     $OP2-1(2)       ; CHANGE SIGN IF FLAG SET
539 1269 CA04           ST      $OP2-1(2)       ; SAVE CORRECTED SIGN OF RESUL
540 126B C205           LD      $OP2(2)         ;CHECK FOR ZERO.
541 126D 9CA6           JNZ     $OUT            ; RETURN
542 126F C206           LD      $OP2+1(2)
543 1271 9CA2           JNZ     $OUT
544 1273 C207           LD      $OP2+2(2)
545 1275 9C9E           JNZ     $OUT
546 1277 C400           LDI     0
547 1279 CA04           ST      $OP2-1(2)       ;ZERO SIGN.
548 127B 9098           JMP     $OUT
549
550             ;
551
```

TABLE D-7. BCD COMPLEMENT

```
                        .PAGE   'BCD COMPLEMENT'
552                     .LOCAL
553             ;
554             ;       BCDCMP: COMPLEMENT A 6-DIGIT BCD NUMBER.
555             ;
556
557
558             ;
559             ;       CALL:   XPPC  1
560             ;       P2 POINTS TO FIRST BYTE OF NUMBER TO BE COMPLEMENTED.
561             ;
562             ;       USED BY BCD ADD/SUBTRACT ONLY
563             ;
564 127D 08     BCDCMP: NOP
565 127E C49A           LDI     09A
566 1280 01             XAE
```

```
567 1281 40             LDE
568 1282 FA02           CAD     2(2)
569 1284 CA02           ST      2(2)
570 1286 02             CCL
571 1287 40             LDE
572 1288 FA01           CAD     1(2)
573 128A CA01           ST      1(2)
574 128C 02             CCL
575 128D 40             LDE
576 128E FA00           CAD     0(2)
577 1290 CA00           ST      0(2)
578 1292 03             SCL
579 1293 C400           LDI     0
580 1295 EA02           DAD     2(2)
581 1297 CA02           ST      2(2)
582 1299 C400           LDI     0
583 129B EA01           DAD     1(2)
584 129D CA01           ST      1(2)
585 129F C400           LDI     0
586 12A1 EA00           DAD     (2)
587 12A3 CA00           ST      (2)
588 12A5 3D             XPPC    1
589             ;
590
```

TABLE D-8. BCD DIVIDE

```
                         PAGE    'BCD DIVIDE'
591                      LOCAL
592             ;
593             ;        BCDDIV: BCD DIVIDE
594             ;
595             ;        OPERANDS ARE 6 DIGITS PLUS SIGN
596             ;
597             ;        EXECUTION TIME: 6 TO 127 MILLISEC.   (35 TYPICAL)
598             ;
599             ;        CALL:   XPPC  3
600             ;                <ERROR RETURN>
601             ;                <NORMAL RETURN>
602             ;
603             ;        STACK USAGE:
604             ;                REL  ENTRY        USE           RETURN
605             ;                ---  ------------ ------------ ------------
606             ;                 -8               TEMP
607             ;                 -7               COUNT
608             ;                 -6               P3(H) SAVE
609             ;                 -5               P3(L) SAVE
610                               -4               QUOTIENT(SIGN)
611             ;                 -3               QUOTIENT(1)
612             ;                 -2               QUOTIENT(2)
613             ;                 -1               QUOTIENT(3)
614             ;           (P2)->  0  DIVISOR(S)  DIVISOR(SIGN)
615             ;                   1  DIVISOR(1)  DIVISOR(1)
616             ;                   2  DIVISOR(2)  DIVISOR(2)
                ;
```

```
                ;                       3  DIVISOR(3)  DIVISOR(3)
                ;     (P2,EXIT)->       4  DIVIDEND(S) DIVIDEND(S) QUOTIENT(S)
                ;                       5  DIVIDEND(1) DIVIDEND(1) QUOTIENT(1)
                ;                       6  DIVIDEND(2) DIVIDEND(2) QUOTIENT(2)
                ;                       7  DIVIDEND(3) DIVIDEND(3) QUOTIENT(3)
                ;
          FFF8  $TEMP     =       -8
          FFF9  $CNT      =       -7
          FFFA  $P3       =       -6
          FFFD  $0        =       -3              ; QUOTIENT
          0001  $D1       =       1               ; DIVISOR
          0005  $D2       =       5               ; DIVIDEND (QUOTIENT UPON RETU
                ;
12A6 08   BCDDIV: NOP
12A7 C201         LD      $D1(2)                  ; CHECK FOR ZERO DIVISOR
12A9 9C0B         JNZ     $C1
12AB C202         LD      $D1+1(2)
12AD 9C07         JNZ     $C1
12AF C203         LD      $D1+2(2)
12B1 9C03         JNZ     $C1
12B3 3F           XPPC    3                       ; ERROR - RETURN
12B4 90F0         JMP     BCDDIV
12B6 37   $C1:    XPAH    3                       ; SAVE P3
12B7 CAFA         ST      $P3(2)
12B9 33           XPAL    3
12BA CAFB         ST      $P3+1(2)
12BC C401         LDI     1                         COUNT = 1
12BE CAF9         ST      $CNT(2)
12C0 C200         LD      0(2)                    ;GET SIGN.
12C2 E204         XOR     4(2)                    ;CHECK IF THE SAME.
12C4 CAFC         ST      -4(2)                   ;SAVE SIGN.
12C6 C201 $SET:   LD      $D1(2)                  ; NORMALIZE THE DIVISOR
12C8 D4F0         ANI     0F0                     ; CHECK FOR ZERO HIGH DIGIT
12CA 9C1F         JNZ     $SETUP                  ;    NOT ZERO - NORMALIZATION
12CC C404         LDI     4
12CE CAF8         ST      $TEMP(2)
12D0 02   $SL1:   CCL                               SHIFT DIVISOR LEFT 1 BIT
12D1 C203         LD      $D1+2(2)                ;    (DONE 4 TIMES)
12D3 F203         ADD     $D1+2(2)
12D5 CA03         ST      $D1+2(2)
12D7 C202         LD      $D1+1(2)
12D9 F202         ADD     $D1+1(2)
12DB CA02         ST      $D1+1(2)
12DD C201         LD      $D1(2)
12DF F201         ADD     $D1(2)
12E1 CA01         ST      $D1(2)
12E3 BAF8         DLD     $TEMP(2)
12E5 9CE9         JNZ     $SL1
12E7 AAF9         ILD     $CNT(2)                 ; COUNT = COUNT + 1
12E9 90DB         JMP     $SET
12EB C400 $SETUP  LDI     0                       ; QUOTIENT = 0
12ED CAFD         ST      $0(2)
12EF CAFE         ST      $0+1(2)
12F1 CAFF         ST      $0+2(2)
12F3 CA00         ST      0(2)                    ;ZERO SIGNS.
```

```
672 12F5 CA04           ST      4(2)
673 12F7 9002           JMP     $SUB                START WITH SUBTRACTION
674              ;
675              ;      INCREMENT QUOTIENT - ILD MAY BE USED SINCE THERE
676              ;      WILL NEVER BE A CARRY FROM 1 DIGIT TO THE NEXT
677 12F9 AAFF  $INC0:  ILD     $0+2(2)             INCREMENT QUOTIENT (BCD)
678 12FB C411  $SUB:   LDI     H(BCDSUB)         ; CALL BCDSUB
679 12FD 37            XPAH    3                 ;    TO DO SUBTRACTION
680 12FE C4E1          LDI     L(BCDSUB)-1       ;    (DIVIDEND-DIVISOR)
681 1300 33            XPAL    3
682 1301 3F            XPPC    3
683 1302 9000          JMP     ONN
684 1304 C204  ONN:    LD      $D2-1(2)          ; IS RESULT POSITIVE?
685 1306 98F1          JZ      $INC0             ;    YES - INCREMENT QUOTIENT
686 1308 C411          LDI     H(BCDADD)         ;    NO - ADD DIVISOR BACK TO
687 130A 37            XPAH    3
688 130B C4F4          LDI     L(BCDADD)-1
689 130D 33            XPAL    3
690 130E 3F            XPPC    3
691 130F 9000          JMP     ON
692 1311 BAF9  ON.     DLD     $CNT(2)             COUNT = COUNT - 1
693 1313 982D          JZ      $DONE               IF COUNT=0 THEN DIVISION IS
694 1315 C404          LDI     4
695 1317 CAF8          ST      $TEMP(2)
696 1319 C201  $SL2:   LD      $D1(2)            ; DIVIDE DIVISOR BY 10
697 131B 02            CCL                       (SHIFT RIGHT 4 BITS)
698 131C 1F            RRL
699 131D CA01          ST      $D1(2)
700 131F C202          LD      $D1+1(2)
701 1321 1F            RRL
702 1322 CA02          ST      $D1+1(2)
703 1324 C203          LD      $D1+2(2)
704 1326 1F            RRL
705 1327 CA03          ST      $D1+2(2)
706 1329 02            CCL                       MULTIPLY QUOTIENT BY 10
707 132A C2FF          LD      $0+2(2)           ;    (SHIFT LEFT 4 BITS)
708 132C F2FF          ADD     $0+2(2)
709 132E CAFF          ST      $0+2(2)
710 1330 C2FE          LD      $0+1(2)
711 1332 F2FE          ADD     $0+1(2)
712 1334 CAFE          ST      $0+1(2)
713 1336 C2FD          LD      $0(2)
714 1338 F2FD          ADD     $0(2)
715 133A CAFD          ST      $0(2)
716 133C BAF8          DLD     $TEMP(2)
717 133E 9CD9          JNZ     $SL2
718 1340 90B9          JMP     $SUB
719              ;
720 1342 C2FC  $DONE:  LD      $0-1(2)           ; COPY QUOTIENT TO
721 1344 CA04          ST      $D2-1(2)          ;    RETURN LOCATIONS
722 1346 C2FD          LD      $0(2)
723 1348 CA05          ST      $D2(2)
724 134A C2FE          LD      $0+1(2)
725 134C CA06          ST      $D2+1(2)
726 134E C2FF          LD      $0+2(2)
```

```
727 1350 CA07           ST      $D2+2(2)
728 1352 C2FA           LD      $P3(2)          ; RESTORE P3 TO ORIGINAL VALUE
729 1354 37             XPAH    3
730 1355 C2FB           LD      $P3+1(2)
731 1357 33             XPAL    3
732 1358 C702           LD      @2(3)           ; INCREMENT P3 BY 2 FOR NORMAL
733 135A C604           LD      @4(2)           ; INCREMENT P2 TO POINT TO QUO
734 135C 3F     $OUT:   XPPC    3               ; RETURN
735 135D C412           LDI     H(BCDDIV)
736 135F 35             XPAH    1
737 1360 C4A6           LDI     L(BCDDIV)
738 1362 31             XPAL    1
739 1363 91FF           JMP     0(1)            ; GO TO BCDDIV
740             ;
741 1365 C2FA   $ERR:   LD      $P3(2)          ; OVERFLOW IN BCDADD OR BCDSUB
742 1367 37             XPAH    3               ;    RESTORE P3 AND RETURN
743 1368 C2FB           LD      $P3+1(2)
744 136A 33             XPAL    3
745 136B 90EF           JMP     $OUT
746             ;
747
748
749       0000            END
```

MEMORY ASSIGNMENTS

BCDADD	11F5	BCDCMP	127D	BCDDIV	12A6
BCDMPY	111D	BCDSUB	11E2	COUNT	FFFD
DADD	1000	DIV	10A8	DNEG	1013
DSUB	1024	M1	FFFE	M2	0005
M3	FFF6	MPY	1037	NEXTD	1170
NO	105D	NOADD	104C	OK	1231
ON	1311	ONN	1304	OP2Z	1218
PR	FFF7	SAME	11FD	SMPY	1061
TEMP	FFF6	$2	1223	$BSUB	11EE
$C1	12B6	$CHK	11F8	$CNT	FFF9
$D1	10B0	$D1	0001	$D2	0005
$DIFF	121E	$DONE	1114	$DONE	1342
$DVGR	1105	$ERR	1365 *	$INC0	12F9
$L1	1147	$L2	1178	$L3	11B1
$LOOP	1043	$LOOP	10D6	$LOOP	113F
$MPY	1077	$MPY2	1084	$MPY3	108F
$MPY4	10A1	$OP1	0001	$OP2	0005
$OUT	11BD	$OUT	1215	$OUT	135C
$OVFL	1217	$P3	FFFA	$PLUS	1265
$0	FFFD	$RTN2	1267	$SAME	1075
$SET	10B7	$SET	12C6	$SETU	10C3
$SETU	12EB	$SL1	12D0	$SL2	1319
$SUB	12FB	$TEMP	FFF8		

APPENDIX E
IMPLEMENTING PROGRAM DELAYS FOR SC/MP

In some applications, programming delays may be required to properly interface SC/MP with other components of the system. Delays of a few microcycles can be implemented by executing one or more NOP commands—each NOP instruction provides a minimum delay of 5-microcycles (undefined 1-byte opcodes) and a maximum delay of 10-microcycles (undefined 2-byte opcodes). The following table shows how the decimal 'displacement' value and the decimal value of 'accumulator' can be used to achieve any delay within the range of 13-to-131,593 microcycles (1 microcycle = 2 T_X, where T_X = 1/Fosc). If the required delay falls between two values in the table, adequate resolution can be obtained by interpolation.

Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC
13	0	00	1555	3	00	5153	10	00	12863	25	00
53	0	20	1595	3	20	5193	10	20	12903	25	20
93	0	40	1635	3	40	5233	10	40	12943	25	40
133	0	60	1675	3	60	5273	10	60	12983	25	60
173	0	80	1715	3	80	5313	10	80	13023	25	80
213	0	100	1755	3	100	5353	10	100	13063	25	100
253	0	120	1795	3	120	5393	10	120	13103	25	120
293	0	140	1835	3	140	5433	10	140	13143	25	140
333	0	160	1875	3	160	5473	10	160	13183	25	160
373	0	180	1915	3	180	5513	10	180	13223	25	180
413	0	200	1955	3	200	5553	10	200	13263	25	200
453	0	220	1995	3	220	5593	10	220	13303	25	220
493	0	240	2035	3	240	5633	10	240	13343	25	240
523	0	255	2065	3	255	5663	10	255	13373	25	255
527	1	00	2069	4	00	7723	15	00	15433	30	00
567	1	20	2109	4	20	7763	15	20	15473	30	20
607	1	40	2149	4	40	7803	15	40	15513	30	40
647	1	60	2189	4	60	7843	15	60	15553	30	60
687	1	80	2229	4	80	7883	15	80	15593	30	80
727	1	100	2269	4	100	7923	15	100	15633	30	100
767	1	120	2309	4	120	7963	15	120	15673	30	120
807	1	140	2349	4	140	8003	15	140	15713	30	140
847	1	160	2389	4	160	8043	15	160	15753	30	160
887	1	180	2429	4	180	8083	15	180	15793	30	180
927	1	200	2469	4	200	8123	15	200	15833	30	200
967	1	220	2509	4	220	8163	15	220	15873	30	220
1007	1	240	2549	4	240	8203	15	240	15913	30	240
1037	1	255	2579	4	255	8233	15	255	15943	30	255
1041	2	00	2583	5	00	10293	20	00	18003	35	00
1081	2	20	2623	5	20	10333	20	20	18043	35	20
1121	2	40	2663	5	40	10373	20	40	18083	35	40
1161	2	60	2703	5	60	10413	20	60	18123	35	60
1201	2	80	2743	5	80	10453	20	80	18163	35	80
1241	2	100	2783	5	100	10493	20	100	18203	35	100
1281	2	120	2823	5	120	10533	20	120	18243	35	120
1321	2	140	2863	5	140	10573	20	140	18283	35	140
1361	2	160	2903	5	160	10613	20	160	18323	35	160
1401	2	180	2943	5	180	10653	20	180	18363	35	180
1441	2	200	2983	5	200	10693	20	200	18403	35	200
1481	2	220	3023	5	220	10733	20	220	18443	35	220
1521	2	240	3063	5	240	10773	20	240	18483	35	240
1551	2	255	3093	5	255	10803	20	255	18513	35	255

Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC
20573	40	00	28283	55	00	35993	70	00	43703	85	00
20613	40	20	28323	55	20	36033	70	20	43743	85	20
20653	40	40	28363	55	40	36073	70	40	43783	85	40
20693	40	60	28403	55	60	36113	70	60	43823	85	60
20733	40	80	28443	55	80	36153	70	80	43863	85	80
20773	40	100	28483	55	100	36193	70	100	43903	85	100
20813	40	120	28523	55	120	36233	70	120	43943	85	120
20853	40	140	28563	55	140	36273	70	140	43983	85	140
20893	40	160	28603	55	160	36313	70	160	44023	85	160
20933	40	180	28643	55	180	36353	70	180	44063	85	180
20973	40	200	28683	55	200	36393	70	200	44103	85	200
21013	40	220	28723	55	220	36433	70	220	44143	85	220
21053	40	240	28763	55	240	36473	70	240	44183	85	240
21083	40	255	28793	55	255	36503	70	255	44213	85	255
23143	45	00	30853	60	00	38563	75	00	46273	90	00
23183	45	20	30893	60	20	38603	75	20	46313	90	20
23223	45	40	30933	60	40	38643	75	40	46353	90	40
23263	45	60	30973	60	60	38683	75	60	46393	90	60
23303	45	80	31013	60	80	38723	75	80	46433	90	80
23343	45	100	31053	60	100	38763	75	100	46473	90	100
23383	45	120	31093	60	120	38803	75	120	46513	90	120
23423	45	140	31133	60	140	38843	75	140	46553	90	140
23463	45	160	31173	60	160	38883	75	160	46593	90	160
23503	45	180	31213	60	180	38923	75	180	46633	90	180
23543	45	200	31253	60	200	38963	75	200	46673	90	200
23583	45	220	31293	60	220	39003	75	220	46713	90	220
23623	45	240	31333	60	240	39043	75	240	46753	90	240
23653	45	255	31363	60	255	39073	75	255	46783	90	255
25713	50	00	33423	65	00	41133	80	00	48843	95	00
25753	50	20	33463	65	20	41173	80	20	48883	95	20
25793	50	40	33503	65	40	41213	80	40	48923	95	40
25833	50	60	33543	65	60	41253	80	60	48963	95	60
25873	50	80	33583	65	80	41293	80	80	49003	95	80
25913	50	100	33623	65	100	41333	80	100	49043	95	100
25953	50	120	33663	65	120	41373	80	120	49083	95	120
25993	50	140	33703	65	140	41413	80	140	49123	95	140
26033	50	160	33743	65	160	41453	80	160	49163	95	160
26073	50	180	33783	65	180	41493	80	180	49203	95	180
26113	50	200	33823	65	200	41533	80	200	49243	95	200
26153	50	220	33863	65	220	41573	80	220	49283	95	220
26193	50	240	33903	65	240	41613	80	240	49323	95	240
26223	50	255	33933	65	255	41643	80	255	49353	95	255

Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC
51413	100	00	59123	115	00	66833	130	00	74543	145	00
51453	100	20	59163	115	20	66873	130	20	74583	145	20
51493	100	40	59203	115	40	66913	130	40	74623	145	40
51533	100	60	59243	115	60	66953	130	60	74663	145	60
51573	100	80	59283	115	80	66993	130	80	74703	145	80
51613	100	100	59323	115	100	67033	130	100	74743	145	100
51653	100	120	59363	115	120	67073	130	120	74783	145	120
51693	100	140	59403	115	140	67113	130	140	74823	145	140
51733	100	160	59443	115	160	67153	130	160	74863	145	160
51773	100	180	59483	115	180	67193	130	180	74903	145	180
51813	100	200	59523	115	200	67233	130	200	74943	145	200
51853	100	220	59563	115	220	67273	130	220	74983	145	220
51893	100	240	59603	115	240	67313	130	240	75023	145	240
51923	100	255	59633	115	255	67343	130	255	75053	145	255
53983	105	00	61693	120	00	69403	135	00	77113	150	00
54023	105	20	61733	120	20	69443	135	20	77153	150	20
54063	105	40	61773	120	40	69483	135	40	77193	150	40
54103	105	60	61813	120	60	69523	135	60	77233	150	60
54143	105	80	61853	120	80	69563	135	80	77273	150	80
54183	105	100	61893	120	100	69603	135	100	77313	150	100
54223	105	120	61933	120	120	69643	135	120	77353	150	120
54263	105	140	61973	120	140	69683	135	140	77393	150	140
54303	105	160	62013	120	160	69723	135	160	77433	150	160
54343	105	180	62053	120	180	69763	135	180	77473	150	180
54383	105	200	62093	120	200	69803	135	200	77513	150	200
54423	105	220	62133	120	220	69843	135	220	77553	150	220
54463	105	240	62173	120	240	69883	135	240	77593	150	240
54493	105	255	62203	120	255	69913	135	255	77623	150	255
56553	110	00	64263	125	00	71973	140	00	79683	155	00
56593	110	20	64303	125	20	72013	140	20	79723	155	20
56633	110	40	64343	125	40	72053	140	40	79763	155	40
56673	110	60	64383	125	60	72093	140	60	79803	155	60
56713	110	80	64423	125	80	72133	140	80	79843	155	80
56753	110	100	64463	125	100	72173	140	100	79883	155	100
56793	110	120	64503	125	120	72213	140	120	79923	155	120
56833	110	140	64543	125	140	72253	140	140	79963	155	140
56873	110	160	64583	125	160	72293	140	160	80003	155	160
56913	110	180	64623	125	180	72333	140	180	80043	155	180
56953	110	200	64663	125	200	72373	140	200	80083	155	200
56993	110	220	64703	125	220	72413	140	220	80123	155	220
57033	110	240	64743	125	240	72453	140	240	80163	155	240
57063	110	255	64773	125	255	72483	140	255	80193	155	255

Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC
82253	160	00	89963	175	00	97673	190	00	105383	205	00
82293	160	20	90003	175	20	97713	190	20	105423	205	20
82333	160	40	90043	175	40	97753	190	40	105463	205	40
82373	160	60	90083	175	60	97793	190	60	105503	205	60
82413	160	80	90123	175	80	97833	190	80	105543	205	80
82453	160	100	90163	175	100	97873	190	100	105583	205	100
82493	160	120	90203	175	120	97913	190	120	105623	205	120
82533	160	140	90243	175	140	97953	190	140	105663	205	140
82573	160	160	90283	175	160	97993	190	160	105703	205	160
82613	160	180	90323	175	180	98033	190	180	105743	205	180
82653	160	200	90363	175	200	98073	190	200	105783	205	200
82693	160	220	90403	175	220	98113	190	220	105823	205	220
82733	160	240	90443	175	240	98153	190	240	105863	205	240
82763	160	255	90473	175	255	98183	190	255	105893	205	255
84823	165	00	92533	180	00	100243	195	00	107953	210	00
84863	165	20	92573	180	20	100283	195	20	107993	210	20
84903	165	40	92613	180	40	100323	195	40	108033	210	40
84943	165	60	92653	180	60	100363	195	60	108073	210	60
84983	165	80	92693	180	80	100403	195	80	108113	210	80
85023	165	100	92733	180	100	100443	195	100	108153	210	100
85063	165	120	92773	180	120	100483	195	120	108193	210	120
85103	165	140	92813	180	140	100523	195	140	108233	210	140
85143	165	160	92853	180	160	100563	195	160	108273	210	160
85183	165	180	92893	180	180	100603	195	180	108313	210	180
85223	165	200	92933	180	200	100643	195	200	108353	210	200
85263	165	220	92973	180	220	100683	195	220	108393	210	220
85303	165	240	93013	180	240	100723	195	240	108433	210	240
85333	165	255	93043	180	255	100753	195	255	108463	210	255
87393	170	00	95103	185	00	102813	200	00	110523	215	00
87433	170	20	95143	185	20	102853	200	20	110563	215	20
87473	170	40	95183	185	40	102893	200	40	110603	215	40
87513	170	60	95223	185	60	102933	200	60	110643	215	60
87553	170	80	95263	185	80	102973	200	80	110683	215	80
87593	170	100	95303	185	100	103013	200	100	110723	215	100
87633	170	120	95343	185	120	103053	200	120	110763	215	120
87673	170	140	95383	185	140	103093	200	140	110803	215	140
87713	170	160	95423	185	160	103133	200	160	110843	215	160
87753	170	180	95463	185	180	103173	200	180	110883	215	180
87793	170	200	95503	185	200	103213	200	200	110923	215	200
87833	170	220	95543	185	220	103253	200	220	110963	215	220
87873	170	240	95583	185	240	103293	200	240	111003	215	240
87903	170	255	95613	185	255	103323	200	255	111033	215	255

Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC	Delay μcycles	Decimal Disp.	Decimal ACC
113093	220	00	120803	235	00	128513	250	00			
113133	220	20	120843	235	20	128553	250	20			
113173	220	40	120883	235	40	128593	250	40			
113213	220	60	120923	235	60	128633	250	60			
113253	220	80	120963	235	80	128673	250	80			
113293	220	100	121003	235	100	128713	250	100			
113333	220	120	121043	235	120	128753	250	120			
113373	220	140	121083	235	140	128793	250	140			
113413	220	160	121123	235	160	128833	250	160			
113453	220	180	121163	235	180	128873	250	180			
113493	220	200	121203	235	200	128913	250	200			
113533	220	220	121243	235	220	128953	250	220			
113573	220	240	121283	235	240	128993	250	240			
113603	220	255	121313	235	255	129023	250	255			
115663	225	00	123373	240	00	131083	255	00			
115703	225	20	123413	240	20	131123	255	20			
115743	225	40	123453	240	40	131163	255	40			
115783	225	60	123493	240	60	131203	255	60			
115823	225	80	123533	240	80	131243	255	80			
115863	225	100	123573	240	100	131283	255	100			
115903	225	120	123613	240	120	131323	255	120			
115943	225	140	123653	240	140	131363	255	140			
115983	225	160	123693	240	160	131403	255	160			
116023	225	180	123733	240	180	131443	255	180			
116063	225	200	123773	240	200	131483	255	200			
116103	225	220	123813	240	220	131523	255	220			
116143	225	240	123853	240	240	131563	255	240			
116173	225	255	123883	240	255	131593	255	255			
118233	230	00	125943	245	00						
118273	230	20	125983	245	20						
118313	230	40	126023	245	40						
118353	230	60	126063	245	60						
118393	230	80	126103	245	80						
118433	230	100	126143	245	100						
118473	230	120	126183	245	120						
118513	230	140	126223	245	140						
118553	230	160	126263	245	160						
118593	230	180	126303	245	180						
118633	230	200	126343	245	200						
118673	230	220	126383	245	220						
118713	230	240	126423	245	240						
118743	230	255	126453	245	255						