8051 Interfacing and Applications

Applied Logic Engineering

System Design and Development

Disclaimer

Every effort has been made to make this manual as accurate and complete as possible. Applied Logic Engineering is not responsible for inaccuracies, omissions, etc. that may have occurred during preparation of this manual or problems,damage, or loss that may result from its use.

Table of Contents

1. 8051 Interfacing and Applications

1.1. Introduction

The purpose of this manual is to aid designers of 8051-family systems by providing simple, straight-forward ways to interface various peripheral subsystems for single board-8051 designs. By adding these features, any system design can be enhanced to provide additional capability and flexibility.

Whether you are designing your own single board computer from the "ground-up" or you are using a commericially available board for the microcontroller core, a number of these peripheral add-ons provide standard additions to your embedded system design. By using these predesigned solutions, you will save many hours of unnecessary hardware and software design and debugging.

This manual is organized into sections that discuss various peripherals that may be interfaced to the 8051 and includes both sample hardware and software for direct implementation. These designs have been implemented in various systems at Applied Logic Engineering and have been proven to work. While they may be implemented directly, you may choose to adapt these items as you see fit for your individual design.

The software provided in this manual has been also provided as source code files on disk. Each listing is provided in a separate ".ASM" file for use in your design.

The software provided here is written to be easily understood by the novice. The goal is to present workable solutions, but with a few hours work, the algorithms can be optimized to execute more efficiently if required.

It is important to understand that this manual is written for users that have a basic understanding of digital design concepts and some understanding of 8051 software design. The assumption that the user will have had some experience with the components described will be made in the cases of standard TTL logic components (i.e. AND, NAND, OR, etc.). If these items are not familiar to you, a reference book describing these components may be required.

An appendix at the end of this manual will give the names of sources of the manufacturers of the components covered in the designs discussed. Also, an appendix is provided that discusses how "outside world" connections can be made between the single board computer and a personal computer.

1.2. Main System Core

1.2.1 Microcontroller and Support Hardware

The core of any microprocessor-based design is the microprocessor itself. This manual will describe designs based around the Intel 8031 microcontroller, but they can easily be adapted to other members of the 8051 family.

The Intel 8051 family of microcontrollers were designed for low cost embedded control systems. These microcontrollers have the capability of direct manipulation of inputs and outputs connected to the 8051.

In addition to direct I/O capability, the 8051 has internal hardware timers that can be used as timers or counters. This provides for capability that normally requires external support chips for normal microprocessor-based designs.

The 8051 family of microcontrollers also has two hardware interrupts included on the chip, eliminating the need for an external interrupt controller in most designs.

Either 128 or 256 bytes of internal RAM are also included on the chip, depending on the model of the chip used.

The 8051 family consists of many derivatives, with some of the most popular being listed below.

8051 Microcontrollers

8051 Internal masked ROM - 128 bytes RAM, two timers

8031 No ROM - 128 bytes RAM, two timers

8751 Internal EPROM - 128 bytes RAM, two timers

8052 Internal masked ROM - 256 bytes RAM, 3 timers

8032 No ROM - 256 bytes RAM, 3 timers

8052BASIC Built-in BASIC language 8052

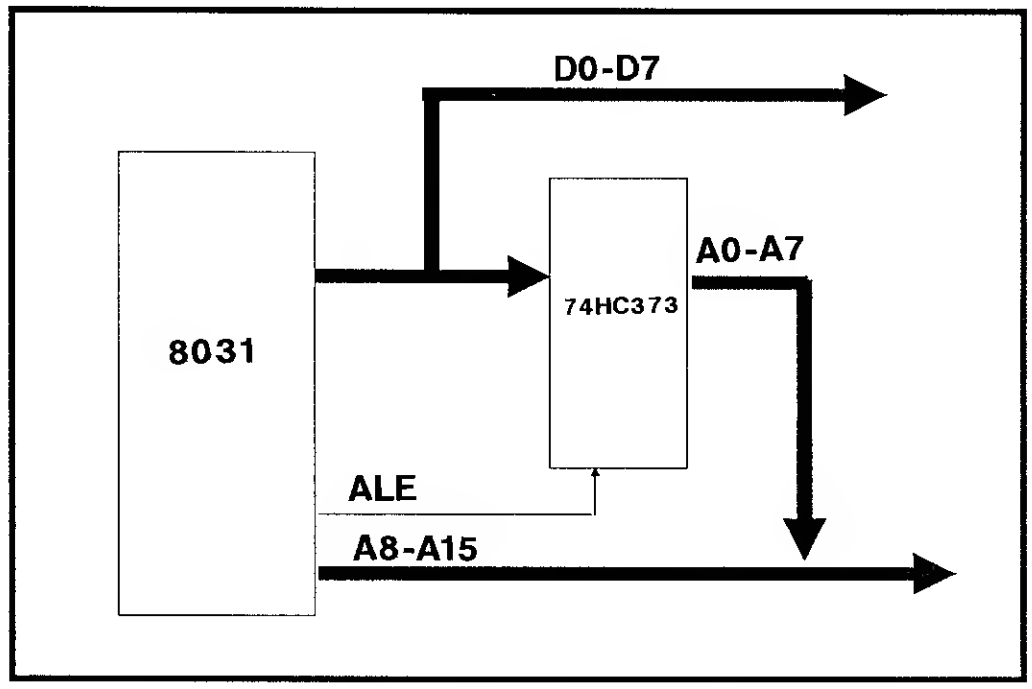
Incorporating an 8031 into a single board computer design is relatively straight forward. The chip can be configured in many different ways, but for the purpose of this discussion, various design decisions have been made and will be discussed in detail. For a detailed discussion on the capabilities of various 8051-type chips, please refer to the Intel literature concerning the individual chip.

First, the oscillator in our sample design is a simple crystal running at 11.0592 MHZ.

This is a standard rate for the 8051 family microcontroller that allows the user to program the internal timers used by the internal UART to standard bits-per-second rates (i.e. 9600, 1200, etc.). Programming the 8031 for this operation is discussed in the section of this manual covering the use of the serial interface that is built into the 8031 itself.

Reset (pin 9) on the 8031 is connected to a 4.7 uF capacitor to provide a simple reset when power is applied to the system.

The *EA (external address, pin 31) is connected to ground to notify the 8031 that the chip does not contain any internal program ROM and that all program code must be available on the external bus. If a chip such as the 8751 is used (which includes internal



Low Order Address Latch

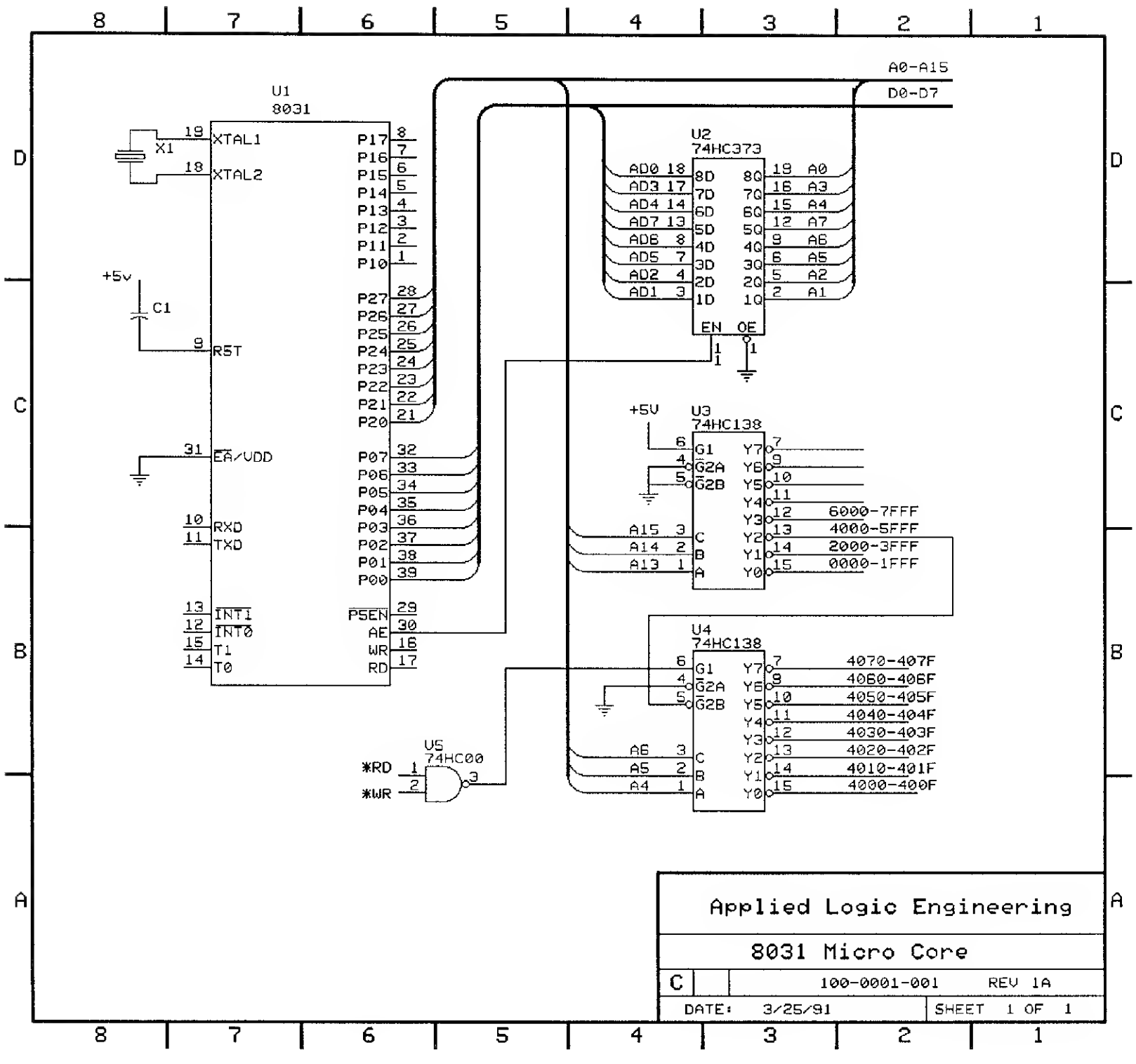
EPROM), this pin must be tied HIGH to allow the 8751 to use the internal ROM for program space.

Grounding the *EA pin requires that P0.0-P0.7 (pins 32-39) are used for the AD0-AD7 signals to form the low order address/data bus and that P2.0-P2.7 (pins 21-28) form the high order address lines A8-A15.

P3.6 (pin 16) becomes the external *WR signal and P3.7 (pin 17) becomes the *RD signal for interfacing to external memory devices. *PSEN provides a chip enable signal for the external ROM that holds the microcontroller program.

The *RD and *WR signals are only active when accessing external data memory. They are not active while doing instruction fetches from ROM, as the following table indicates:

	*PSEN	*RD	*WR
Program Instruction Fetch	0	1	1
External Data Read	1	0	1
External Data Write	1	1	0


A0-A15
D0-D7
U1
8031
XTAL1
XTAL2
X1
+5v
C1
RST
EA/VDD
RXD
TXD
INT1
INT0
T1
T0
P17
P16
P15
P14
P13
P12
P11
P10
P27
P26
P25
P24
P23
P22
P21
P20
P07
P06
P05
P04
P03
P02
P01
P00
PSEN
AE
WR
RD
U2
74HC373
AD0
AD3
AD4
AD7
AD6
AD5
AD2
AD1
8D
7D
6D
5D
4D
3D
2D
1D
8Q
7Q
6Q
5Q
4Q
3Q
2Q
1Q
EN
OE
A0
A3
A4
A7
A6
A5
A2
A1
+5V
U3
74HC138
G1
G2A
G2B
A15
A14
A13
C
B
A
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0
6000-7FFF
4000-5FFF
2000-3FFF
0000-1FFF
U4
74HC138
4070-407F
4060-406F
4050-405F
4040-404F
4030-403F
4020-402F
4010-401F
4000-400F
U5
74HC00
*RD
*WR
Applied Logic Engineering
8031 Micro Core
C
100-0001-001
REV 1A
DATE: 3/25/91
SHEET 1 OF 1

Any other pins not designated above can be used for general purpose I/O or for specific, pre-defined purposes, depending on the individual design. Please refer to the Intel literature that describes the definition of these functions.

Low order Address Latch

As most users of Intel microcontrollers and microprocessors are aware, Intel provides a multiplexing system that uses the P0.0-P0.7 lines for both the data bus and the low order byte of the address bus. In order for the system to interpret these signals, an external latch must be used to separate the address information from the data information. In this design, a 74HC373 latch is used. The ALE (Address Latch Enable, pin30) signal on the 8031 provides the enable for the 74HC373 to latch the address information onto the bus.

Memory Decoding

In order for the 8031 to find and execute the software it was intended to run, some sort of ROM is usually provided at address 0000h (for the RESET vector). Other interrupt vectors have their address locations in the area of memory from 0003h-002Bh.

In the implementation described in this manual, EPROM will be mapped at 0000h-1FFFh and also at 2000h-3FFFh. This provides for 16K bytes of program space for the operating system software.

A simple method of memory decoding for memory devices is designed into the system by including a 74HC138 3-of-8 decoder. By using address lines A13, A14, and A15, the 74HC138 decodes output signals in 8K blocks. This will be adequate for this design, but you may choose to use larger or smaller memory blocks, depending on your require-ments.

In the design presented, the following table shows the method in which the memory decoding is done using the A13, A14, and A15 address linesto give the 8K block outputs

Assuming the inputs on the 74HC138 are configured:
G1 = 1, *G2A = 0, and *G2B = 0

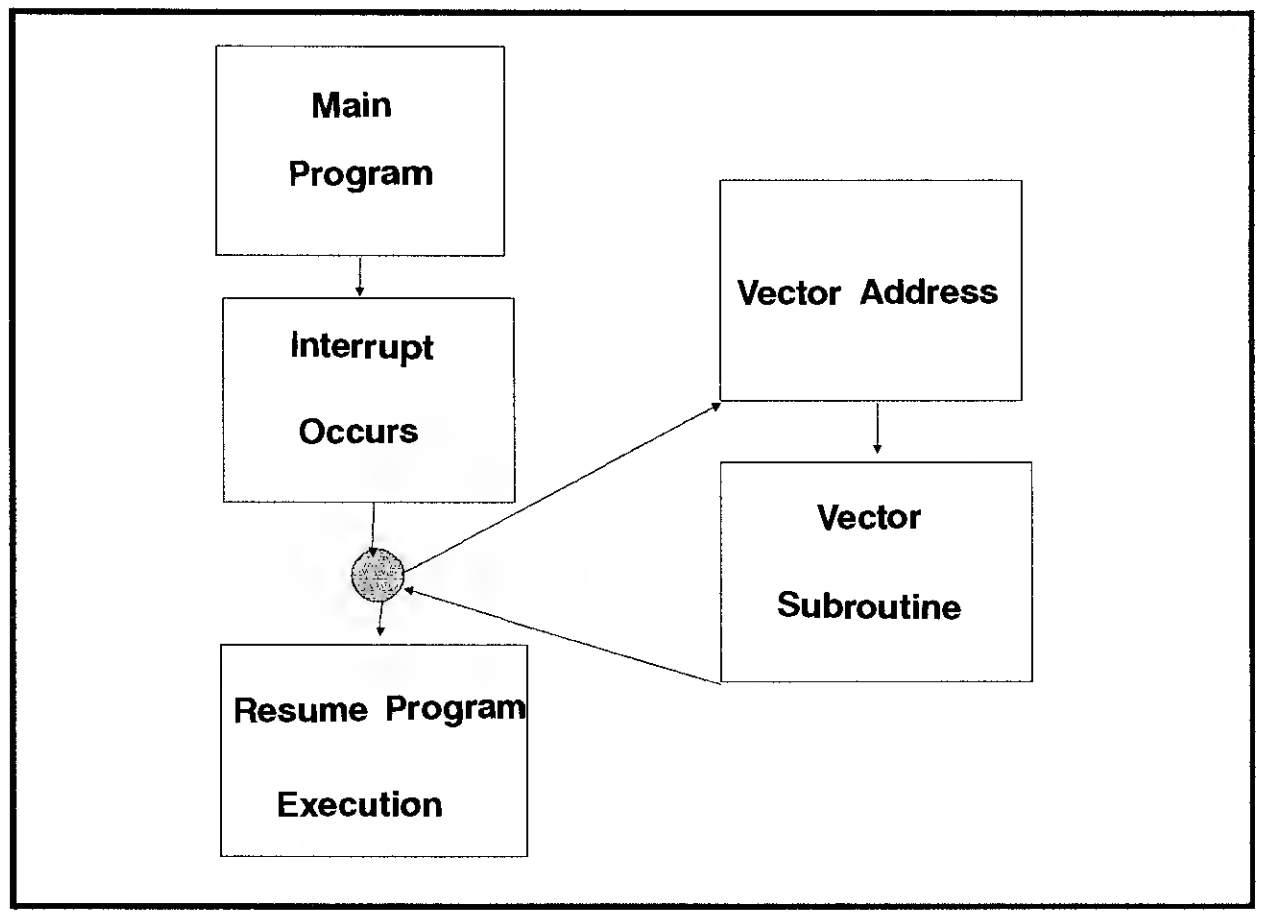
the output table would then be:

A15	A14	A13	*Y0	*Y1	*Y2	*Y3	*Y4	*Y5	*Y6	*Y7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

A second 74HC138 has been included to provide a smaller block decode within one of the 8K blocks for interfacing to various I/O devices. The output of this decoder provides blocks of 16 bytes each. Note that due to the fact that this decoder uses *RD and *WR from the 8031 for enabling its outputs, it can only be used for enabling external memory devices.

Interrupts

The 8031 internally provides for two hardware interrupts (INT0 and INT1) that can be used without additional support circuitry. In most single board computer designs, interrupts are used for connecting the system to devices that require immediate response service and that can occur without predictability. Some examples of these types of devices include serial UARTS(where data is received in an unpredictable stream),



parallel input data, direct switch contacts, etc.

Another advantage to the use of interrupts is the elimination of software input "polling" by the main operating system. Polling is the process by where the software periodically reads a particular input or group of inputs to determine if an event had occured or not. This can lead to system overhead problems if the inputs need to be polled very frequently in order to not miss an event that could happen on the input itself. Quite a bit of CPU time can be expended by simply completing the input polling. By going to an interrupt-based input, the input event will cause the main software to stop and service the event. All overhead in reading the input in a periodic form is eliminated and the system will never "miss" an input event.

By programming the internal registers of the 8031, you can choose to use the *INT0/P3.2 pin as an interrupt input or as a general purpose I/O bit. The same is true for the *INT1/P3.3 pin.

Also included in the 8031 are two internal hardware timers (Timer 0 and Timer 1) that can be configured to interrupt when the respective timer overflows. Each timer can be configured as a free running timer or as a counter to count transitions on its input pin (T0/P3.4 for timer 0 and T1/P3.5 for timer 1).

In this manual, various examples will be shown using the hardware interrupts for different purposes to give you an idea of the types of things that can be done with them.

1.2.2 Software for Microcontroller Core

Internal RAM Use

The 8031 has 128 bytes of RAM built-in to the chip itself. This RAM is used for several purposes, which will be described below:

1) The 8031 has a defined group of general use registers named R0-R7. These registers are duplicated four times, once in each of Register Banks 0-3. When the 8031 powers up, Bank 0 is selected as the default for use, so register R0 appears at internal address 00h, register R1 appears at address 01h, and so on up to register R7 at address 07h.

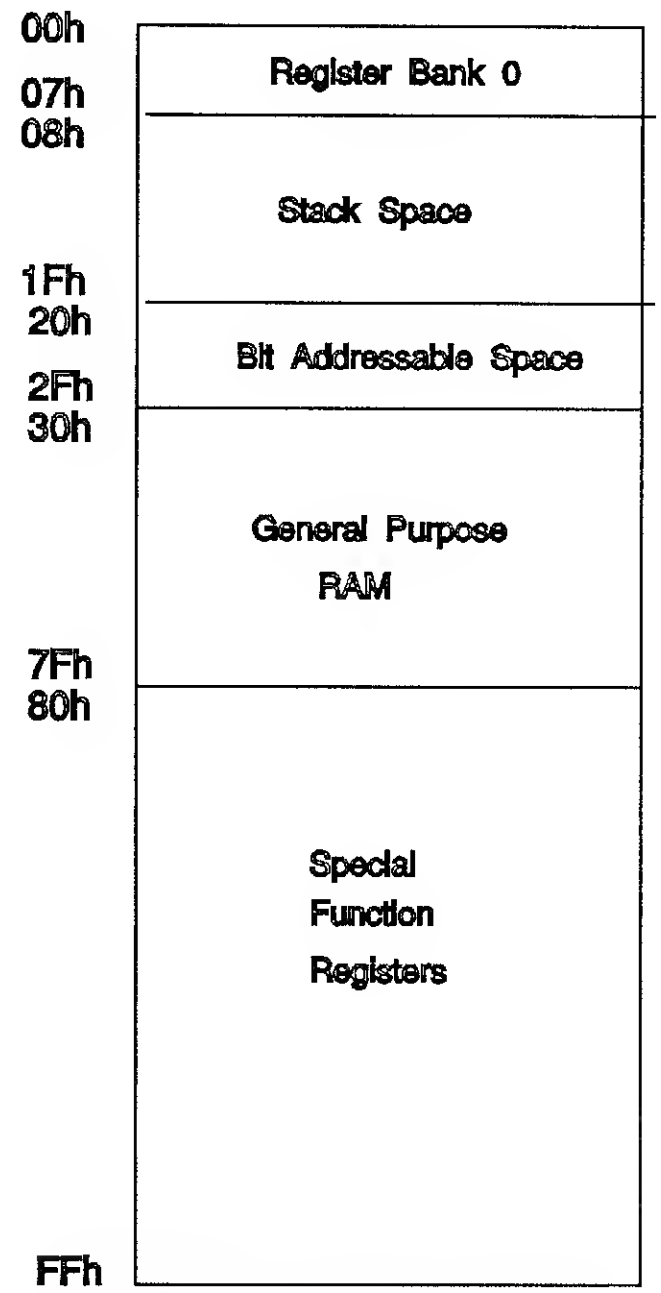
2) Also at power up, the stack pointer is initialized to 07h and incremented to 08h. This default condition assumes register bank 0 will be used for the R0-R7 registers. Also, banks 1-3 for these registers will not be available because this area will be used as the stack grows during use. If all of this memory is assumed to be used by the stack, it will extend from 08h to 1Fh. If your application will make use of any register bank other than Bank 0, it will be necessary to reprogram the stack pointer to an area of memory that will not be affected by any other program operation.

3) Beginning at 20h, the 8031 provides for sixteen bytes of bit-addressable memory. This means that the memory in this area can be addressed as individual bits for program flag

usage. This area extends from 20h-2Fh.

4) The 8031 provides for scratch pad RAM in the area from 30h-7Fh. This is RAM for use in general purpose variable storage. It is byte addressable.

5) The area of RAM from 80h-FFh is allocated for use by the Special Function registers. These registers include the internal working registers of the 8031, such as the accumulator, PSW, B, and DPTR registers as well as the registers such as SCON, SBUF, TCON, TMOD, etc. which control the function of the chip itself.



Internal RAM Allocation in an 8031

It is important to note that although not every byte in this block of memory is assigned a register function in the 8031, the unused byte locations cannot be used as general purpose RAM because the design of the chip will not allow it. Intel has reserved these extra memory locations for future use in other derivitives of the 8051-based family.

One item worth noting as far as internal memory goes is the fact that the 8052 derivitives of the 8051 family (including the 8052, 8032, and 8052AHBASIC) have 256 bytes of internal RAM available. The additional 128 bytes of memory exist in the range from 80h-FFh. To avoid conflict with the Special Function Register area (which is memory mapped to the same locations), the software can only access the extended memory by using indirect addressing modes. Sample software instructions to read a byte of data

into the accumulator from location 80h would be:

```
MOV     R0,#80h
MOV     A,@R0
```

Using these types of software instructions, confusion is avoided between this area of memory and the Special Function register area.

External Memory Addressing

If there is any additional read/write memory or devices that are included in the system (in addition to the internal memory of the 8031), it must be accessed using indirect addressing schemes. A common way of doing this is through the use of the DPTR sixteen bit register, which has the ability to hold an entire sixteen bit address. For example:

```
MOVX    A,@DPTR
```

reads the byte of data into the accumulator from the address that the DPTR register is pointing to. In a similar manner:

```
MOVX    @DPTR,A
```

writes the contents of the accumulator to the address held in the DPTR register.

In these cases, the DPTR register must be loaded with the proper address before the MOVX instruction can be executed. Intel has provided a quick way of loading this register using the "MOV DPTR,#data" instruction, where "data" represents any 16 bit immediate value.

Reading from Code Space

The last area of memory that can be accessed by the software is the program memory area. The 8031 has an instruction that uses the DPTR or PC registers to provide a base address that points to the code space. The "A" register can be used to hold an offset that is added to the base address.

```
MOVC    A,@A+DPTR  and
MOVC    A,@A+PC
```

are the two variations of this instruction, the first using the DPTR as the base address and the second using the PC as the base address.

These instructions are useful for reading data from ROM tables. By reading the data value and incrementing the value in the "A" register, a simple loop can be used to access data from a table for processing.

When a valid reset is applied to the 8031, the chip automatically starts program execution at program address 0000h. The system software should be designed to have a "jump" instruction to the main startup code at this address. This is called the RESET VECTOR and is the method that the 8031 uses to begin program execution. The reset vector usually consists of a JMP (or LJMP) instruction, followed by a program address label where the main startup code is located.

The other vector locations in this area from 0000h-002Bh have fixed postion in locations based on the interrupt type. The table below shows the interrupt types and their associated vector position.

Interrupt	Address
External Interrupt 0	0003h
Timer 0	000Bh
External Interrupt 1	0013h
Timer 1	001Bh
Receive and Transmit	0023h
Timer 2 (not in 8031)	002Bh

JMP instructions should be placed in the source code at the vector addresses that have interrupts that will be used in your system. For interrupts that will not be used, it is not important to have JMPs to any particular program address label.

The main program (located at the program address label defined in the Reset Vector) usually consists of setup instructions for programming the 8031 for correct operation. This may include interrupt use, timer use, serial port use, or any direct output pin manipulation that is required by the design itself to set initial conditions on the board.

In the case of the 8031, the P2 outputs (pins 21-28) must be cleared to zero before any external memory reads or writes occur. The reason for this is that this port forms the high order address byte (A8-A15) of the address bus during external memory access, so this sets the state of these lines to a known condition.

The code listing following provides a generic "skeleton" for a basic 8031 program written in assembler. It sets up the basic structures for the data defintion area, the code vector table, and the vector service routines that are called from the vector table.

It also provides code in the startup area for programming the internal registers of the 8031 to configure it for the operation that will be required. The instructions are included to program the required registers, but it will be up to you to set the correct data in these instructions to get the proper results.

1.2.3 Power Supply Requirements

A single +5V power supply is all that is required to power this microcontroller core and any of the designs that will be described in this manual.

Some of the designs presented require voltages other than +5V, but they will be derived with additional components in the design that are powered by the single +5V supply.

This concludes the discussion of the microcontroller and basic system construction.

```
;*****************************************
; 8031 startup skeleton shell
; Copyright 1991 Applied Logic Engineering
;
; may be used without royalty if proper credit ID is
;indicated

;*****************************************
;

          .DATA
;-----------------------------------------
;Addressable bit declarations

Bit0:          REG  20H.0          ;sample - bit 0
;[add other bit declarations here]

;-----------------------------------------
          ORG  30H
;Internal RAM scratchpad area from 30h-7fh

SAMPLE:        REG  30h        ;sample byte at 30h
;[add other RAM variables here]

;-----------------------------------------

          .CODE
;-----------------------------------------
;  VECTOR TABLE
; note : "LJMP  INIT" is the only required vector-all others
;        optional and may be commented out or removed.


          ORG  0000H
          LJMP INIT      ;-jumps to INIT on powerup

          ORG  0003H
          LJMP EXINT0          ;External INT0 vector

          ORG  000BH
          LJMP TIMER0          ;Timer 0 vector

          ORG  0013H
          LJMP EXINT1          ;External INT1 vector

          ORG  001BH
          LJMP TIMER1          ;Timer 1 vector

          ORG  0023H
          LJMP SERIAL          ;Internal UART vector
;-----------------------------------------------

          ORG  40H
;START OF PROGRAM SERVICE ROUTINES
```

```
INIT:

;CLEAR A8-A15 ADDRESS LINES
          CLR  A
          MOV  P2,A
;INITIALIZE TIMERS TO ZERO
          MOV  TL0,#00H
          MOV  TH0,#00H
          MOV  TL1,#00H
          MOV  TH1,#00H
;CONFIGURE TIMER OPERATION
          MOV  TMOD,#00H
;SET INTERRUPT PRIORITY
          MOV  IP,#00000000B
;SET TIMERS RUNNING
          MOV  TCON,#00000000B
;ENABLE INTERRUPTS
          MOV  IE,#00H


;    [other startup and main code goes here]


          JMP  $                ; End of Main line software

;-----------------------------------------------

EXINT0:
          PUSH PSW
          PUSH ACC

;    [external int 0 service routine goes here]

          POP  ACC
          POP  PSW
          RETI

;-----------------------------------------------

TIMER0:
          PUSH PSW
          PUSH ACC

;    [Timer 0 service routine goes here]

          POP  ACC
          POP  PSW
          RETI

;-----------------------------------------------

EXINT1:
          PUSH PSW
```

```
            PUSH ACC

;     [external int 1 service routine goes here]

            POP  ACC
            POP  PSW
            RETI

;-------------------------------------------------------------

TIMER1:
            PUSH PSW
            PUSH ACC

;     [Timer 1 service routine goes here]

            POP  ACC
            POP  PSW
            RETI

;-------------------------------------------------------------

SERIAL:
            PUSH PSW
            PUSH ACC

;     [Serial port service routines go here]

            POP  ACC
            POP  PSW
            RETI

;-------------------------------------------------------------

            END
```

1.3. Simple Methods of User Input

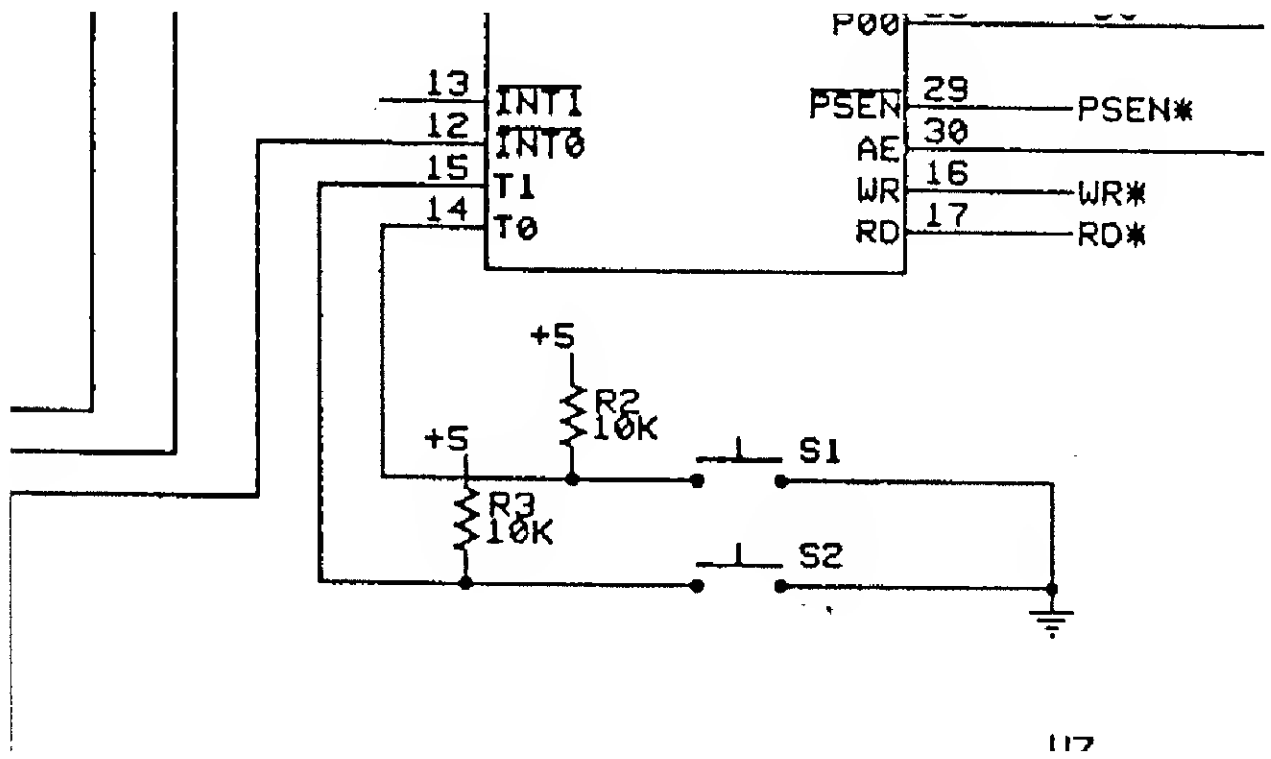
If a simple user interface is required in the design to allow the user to start or stop a particular process in the operating system software, momentary contact switches can be connected directly to any available input pin on the 8031.

One easy way to do this is to provide a normally "open" switch connected to the input pin on one side and a ground connection on the other side. On the input side, a pull-up resistor is included to make the signal appear as a "high" to the 8031 while the switch is not closed. When the switch is pressed, the input will transition from a "high" state to a "low" state at the input pin.

The software can then poll this input periodically to read the state of the input and then can take proper action.

A switch arrangement such as this can be connected on any available port pin (P0.0-P0.7, P1.0-P1.7, P2.0-P2.7, or P3.0-P3.7).

This simple design can be expanded to include multiple switches, or changed to provide



a keyboard scan system for reading switch matrix keyboards or keypads. An example of a more complicated interface will be described in the following section covering interface to multi-digit keypads.

1.3.1 Software

The software that is used to examine the state of the inputs if hardware is implemented as described above is simple. The 8031 provides commands for examining the state of any input pin in the P1, P2, or P3 groups. For example, the software instructions:

```
MOV     C,P3.4

JNC     dosomething

JMP      doanother
```

moves the state of the P3.4 pin (either "0" or "1") into the carry flag. Once this is done, the software can test the state of the carry flag with the "JNC" instruction and take the appropriate action. In the above example, if the state of the carry flag is zero (indicating that the switch connected to P3.4 is currently being pressed), the "JNC" instruction will cause the 8031 to jump to the routine called "dosomething". If the state of the carry flag is high, the switch connected to the P3.4 input is not currently being pressed and the 8031 will execute the routine called "doanother".

Using simple methods such as this, the 8031 can respond to various user-defined input devices.

1.4. Interfacing a 16 digit keypad to the 8031

If a more complex user input is required, a switch matrix arrangement can be designed to interface with the 8031. Basically, the interface is designed to still look for switch closures that indicate that the user has pressed a key, but instead of using individual inputs for each and every switch, multiple switches are connected to the same input bit and decoded using various outputs from the 8031.

The output bits are arranged in "columns" and the input bits in "rows". When a particular column output is turned to a low (0) state, reading the state of the inputs will allow the software to determine if the switch at the intersection of each row connected to that column is closed.

Each column is activated in sequence, with the row inputs being read.

1.4.1 Hardware

The example presented shows the interface to a standard 16 key keypad, which provides for numbers 0-9 plus additional keys that may be used for function keys, an "enter" key, etc.

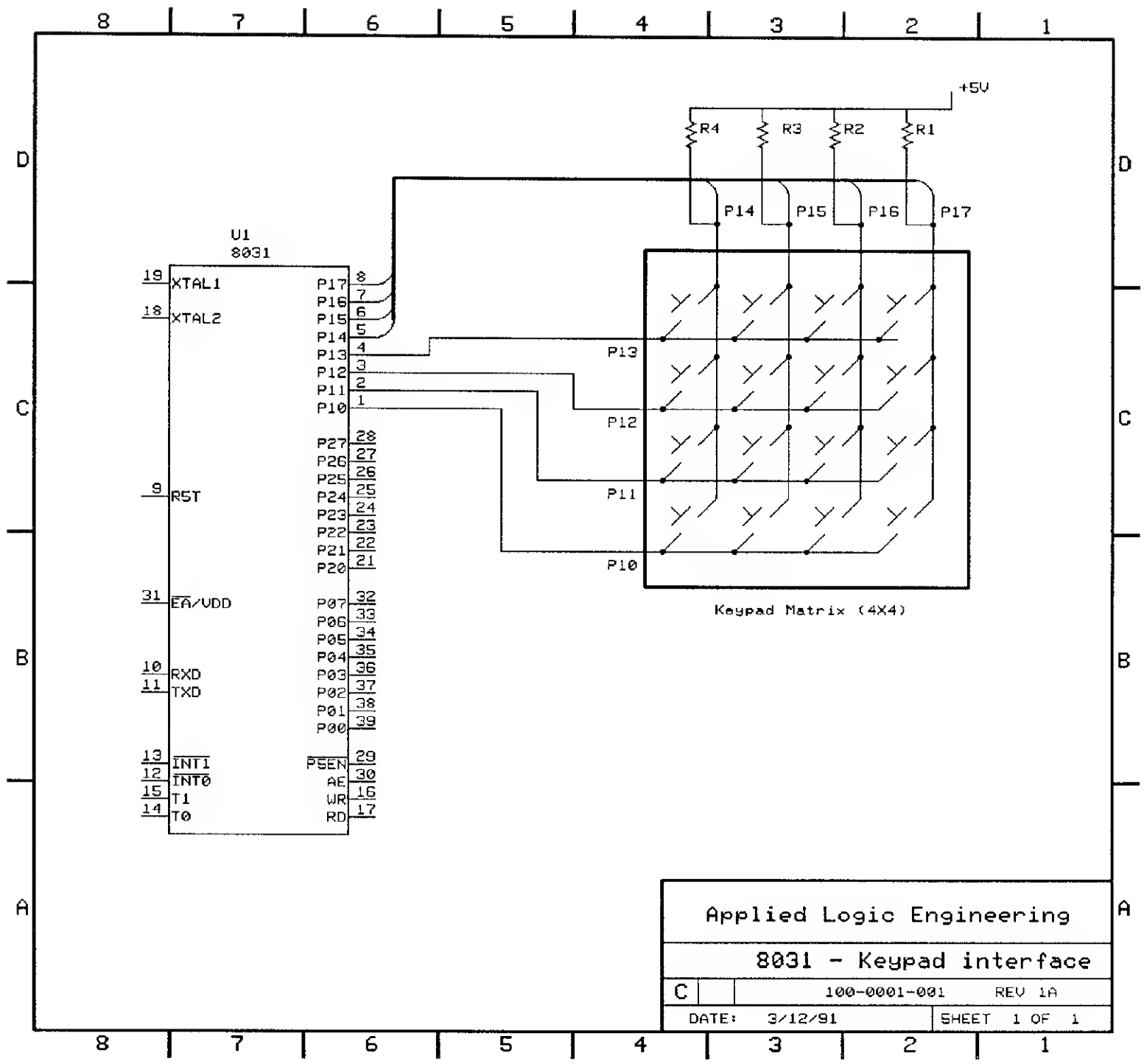
The keypad selected for this sample design organizes the keys into a 4x4 matrix with 4 columns and with 4 rows. The switches are located at the intersections of these rows and columns.

Port 1 on the 8031 is used for the interface to the keypad. Pins P10 ,P11, P12, and P13 are used as the column outputs and P14, P15, P16, and P17 are used for the row inputs. When a column is to be scanned, its corresponding output bit would be brought to a "low" state with the other column output bits being at a "high" state. A very simple hardware interface was designed that "pulls up" the inputs on the 8031 (P14-P17) so that the inputs appear "high" unless the corresponding switch is pressed.

1.4.2 Software

The software designed to be used with this design scans each column/row combination until a valid input is found. At that point, the scan routine will lookup the key value in the ROM table. The software will then execute a jump to a processing routine to handle the action caused by the key closure.

The software also includes a routine for debouncing the key to avoid false "on-off-on"


+5V
R4
R3
R2
R1
P14
P15
P16
P17
U1
8031
XTAL1
XTAL2
RST
EA/VDD
RXD
TXD
INT1
INT0
T1
T0
P17
P16
P15
P14
P13
P12
P11
P10
P27
P26
P25
P24
P23
P22
P21
P20
P07
P06
P05
P04
P03
P02
P01
P00
PSEN
ALE
WR
RD
P13
P12
P11
P10
Keypad Matrix (4X4)
Applied Logic Engineering
8031 - Keypad interface
C
100-0001-001 REV 1A
DATE: 3/12/91
SHEET 1 OF 1

readings by waiting a brief period of time after sensing an "on" state of one of the bits in the row, then rereading the row and making sure that the input is still in a "true" state.

This software implements a "polling" technique as implemented from the main processing software. Depending on the design, a better implementation may be made by going to an "interrupt driven" scheme, where the keyboard scan of all column/row combinations is done based on a periodic interrupt that is triggered from an external source. Usually, a good source for this type of system is an internal timer within the 8031 that can be configured to interrupt the 8031 at a standard programmable rate. The debounce routine can be changed to take advantage of the time between interrupts instead of an artificial delay loop executed by the 8031.

This type of matrix design can be expanded to larger numbers of keys and can be used to do full alphanumeric keyboards for complete user input.

```
;*****************************************
; SOFTWARE TO SCAN A 4X4 MATRIX KEYPAD
;*****************************************


          .CODE
;BEGINNING OF CODE

;*****************************************
;THIS ROUTINE SCANS EACH OF THE 4 OUTPUT COLUMNS
; AND READS KEY CLOSURES FOR THAT COLUMN
;  P1.0-P1.3 are used for the output columns
;  P1.4-P1.7 are used for the input rows
;   USES - R1-TO HOLD PREV BIT DATA
;        R5-KEY COUNT INDEX
;        R3-LOOP COUNTER
;        R6-TEMP
;*****************************************

KEYPAD    EQU  $
          MOV  R1,#20H              ;PREV BIT REG
          MOV  R5,#00H              ;KEY COUNT INDEX
          MOV  R3,#00H              ;LOOP COUNTER

          SETB P1.0                 ;SETUP FOR 1ST COLUMN
          SETB P1.1
          SETB P1.2
          CLR  P1.3

SCAN:     MOV  A,@R1                ;GET CURR "PREV DATA"
          MOV  R6,A
          MOV  R4,#FFH              ;KEY VALUE DEFAULT
          MOV  A,P1                 ;READ INPUTS
          MOV  R2,A                 ;SAVE FOR LATER
          CPL  A
          ANL  A,#80H               ;1ST ROW CHECK
          JZ   KEYS1

          CALL WAIT                 ;DEBOUNCE
          MOV  A,P1                 ;READ DATA AGAIN
          CPL  A
          ANL  A,#80H
          JZ   KEYS1

          MOV  A,R6                 ;PREV BIT ON?
          RRC  A
          JC   KEYS2

          SETB C                    ;NO -SET IT
          RLC  A
          MOV  @R1,A

          MOV  DPTR,#KEYB           ;LOOKUP KEY VALUE
          MOV  A,R5
```

```
                MOVC A,@A+DPTR
                MOV  R4,A                       ;SAVE IT
                LJMP PROCESS

KEYS1:          MOV  A,R6                       ;CLEAR "PREV" BIT
                ANL  A,#FEH
                MOV  R6,A
                MOV  @R1,A

KEYS2:          INC  R5                 ;BUMP KEY COUNTER
                MOV  A,R2               ;CHECK NEXT ROW
                CPL  A
                ANL  A,#40H
                JZ   KEYS3

                CALL WAIT                       ;DEBOUNCE
                MOV  A,P1                       ;GET INPUT AGAIN
                CPL  A
                ANL  A,#40H
                JZ   KEYS3

                MOV  A,R6               ;PREV BIT ON?
                RRC  A
                RRC  A
                JC   KEYS4

                SETB C                  ;NO- SET IT.
                RLC  A
                RLC  A
                MOV  @R1,A

                MOV  DPTR,#KEYB                 ;LOOKUP CHAR VALUE
                MOV  A,R5
                MOVC A,@A+DPTR
                MOV  R4,A
                LJMP PROCESS

KEYS3:          MOV  A,R6               ;CLEAR PREV BIT
                ANL  A,#FDH
                MOV  R6,A
                MOV  @R1,A

KEYS4:          INC  R5                 ;CHECK NEXT ROW
                MOV  A,R2
                CPL  A
                ANL  A,#20H
                JZ   KEYS5

                CALL WAIT               ;DEBOUNCE
                MOV  A,P1               ;GET INPUT AGAIN
                CPL  A
                ANL  A,#20H
                JZ   KEYS5
```

```
        MOV   A,R6
        RRC   A
        RRC   A
        RRC   A
        JC    KEYS6

        SETB  C                 ;SET PREV BIT
        RLC   A
        RLC   A
        RLC   A
        MOV   @R1,A

        MOV   DPTR,#KEYB            ;LOOKUP CHAR VALUE
        MOV   A,R5
        MOVC  A,@A+DPTR
        MOV   R4,A
        LJMP  PROCESS

KEYS5:  MOV   A,R6              ;CLR PREV BIT
        ANL   A,#FBH
        MOV   R6,A
        MOV   @R1,A

KEYS6:  INC   R5                ;CHECK NEXT ROW
        MOV   A,R2
        CPL   A
        ANL   A,#10H
        JZ    KEYS7

        CALL  WAIT              ;DEBOUNCE
        MOV   A,P1
        CPL   A
        ANL   A,#10H
        JZ    KEYS7

        MOV   A,R6              ;PREV BIT SET?
        RRC   A
        RRC   A
        RRC   A
        JC    KEYS8

        SETB  C                 ;NO - SET IT.
        RLC   A
        RLC   A
        RLC   A
        RLC   A
        MOV   @R1,A

        MOV   DPTR,#KEYB            ;LOOKUP KEY VALUE
        MOV   A,R5
        MOVC  A,@A+DPTR
        MOV   R4,A
        LJMP  PROCESS
```

```
KEYS7:      MOV   A,R6              ;CLR PREV BIT
            ANL   A,#F7H
            MOV   @R1,A

KEYS8:      INC   R5                ;DONE WITH SCAN
            MOV   A,R3              ;INC COLUMN COUNT
            INC   A
            MOV   R3,A
            CJNE  A,#01,KEYS9           ;CHECK COL2 NEXT?
            SETB  P1.0              ;YES
            SET   P1.1
            CLR   P1.2
            SET   P1.3
            MOV   R1,#21H
            LJMP  SCAN

KEYS9:      CJNE  A,#02,KEYS10          ;CHECK COL3 NEXT?
            SETB  P1.0              ;YES
            CLR   P1.1
            SETB  P1.2
            SETB  P1.3
            MOV   R1,#22H
            LJMP  SCAN

KEYS10:     CJNE  A,#03,KEYS11          ;CHECK COL4 NEXT?
            CLR   P1.0              ;YES
            SETB  P1.1
            SETB  P1.2
            SETB  P1.3
            MOV   R1,#23H
            LJMP  SCAN
;*****************************************
PROCESS         EQU   $

;     KEY PROCESSING GOES HERE

            RET

;*****************************************************
;SUBROUTINE AREA
;*****************************************************

WAIT        EQU   $
            PUSH  B
            MOV   B,#20H
WAIT1:      NOP                 ;ADJUST TO GET PROPER DELAY
            NOP
            NOP
            DJNZ  B,WAIT1
```

```
            POP   B
            RET


KEYB        EQU   $
            DB    30H         ;"0" CHAR ON KEYPAD
            DB    34H         ;"4"
            DB    38H         ;"8"
            DB    04H         ;F1
            DB    31H         ;"1"
            DB    35H         ;"5"
            DB    39H         ;"9"
            DB    03H         ;F2
            DB    32H         ;"2"
            DB    36H         ;"6"
            DB    2EH         ;"."
            DB    02H         ;F3
            DB    33H         ;"3"
            DB    37H         ;"7"
            DB    00H         ;ENTER
            DB    01H         ;F4
```

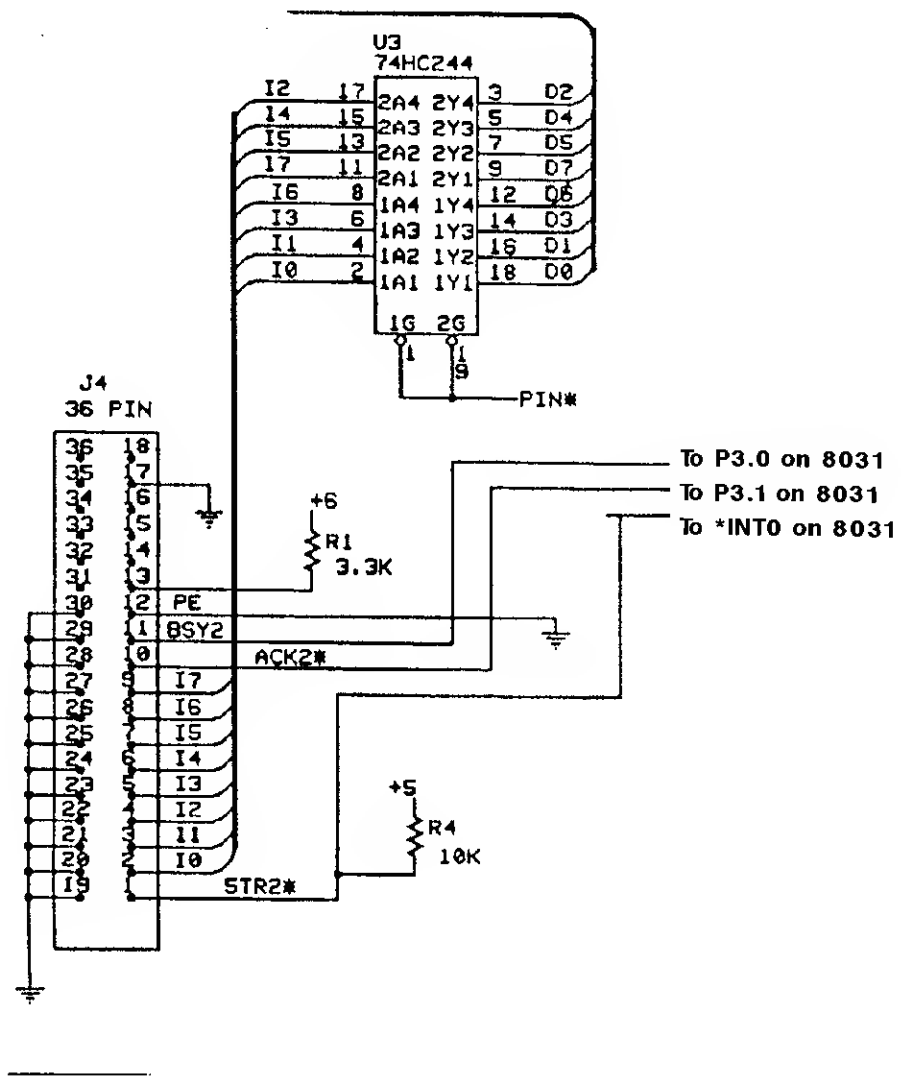
1.5. Centronics Parallel Input Port

This parallel port corresponding to the "Centronics" standard provides for an interface to other computer equipment, such as personal computers.

The interface is an eight bit data input that is timed by an input strobe into the 8031. The input strobe is provided by the computer that is sending the data. By loading the eight bits of data and pulsing the strobe signal, data can be transferred between systems at a very high rate of speed, one byte at a time.

1.5.1 Hardware

The only hardware required for this interface (in addition to a "Centronics"-style 36 pin



connector) is a 74HC244 input buffer. The eight bits of data coming from the 36 pin connector are fed into the input side of the 74HC244, with the outputs being connected to the data bus of the 8031. The enable signal for the 74HC244 is connected to one of the output pins from the second 74HC138 address decoder in the microcontroller core, which provided address decoding in 16 byte blocks.

The *STR signal coming from the transmitting computer over the 36 pin connector can be tied to an interrupt input on the 8031. This line should be pulled up using a 10K ohm resistor connected to +5V to provide the proper state while the *STR signal is in the "OFF" state. The interrupt signal going low then indicates to the 8031 that a byte of data is available to be read on the input port.

The PE (paper end) output from the Centronics connector must be tied to ground to avoid a false "TRUE" to the sending device. If the sending device does see a "TRUE" on this signal, it will believe that the receiving device is "out of paper" (a condition that can occur if the receiving device is a printer), and will not send any additional bytes of data.

1.5.2 Software

The software designed to work with this hardware configuration consists of the vector interrupt service routine for the interrupt input used to connect the *STR input from the connector.

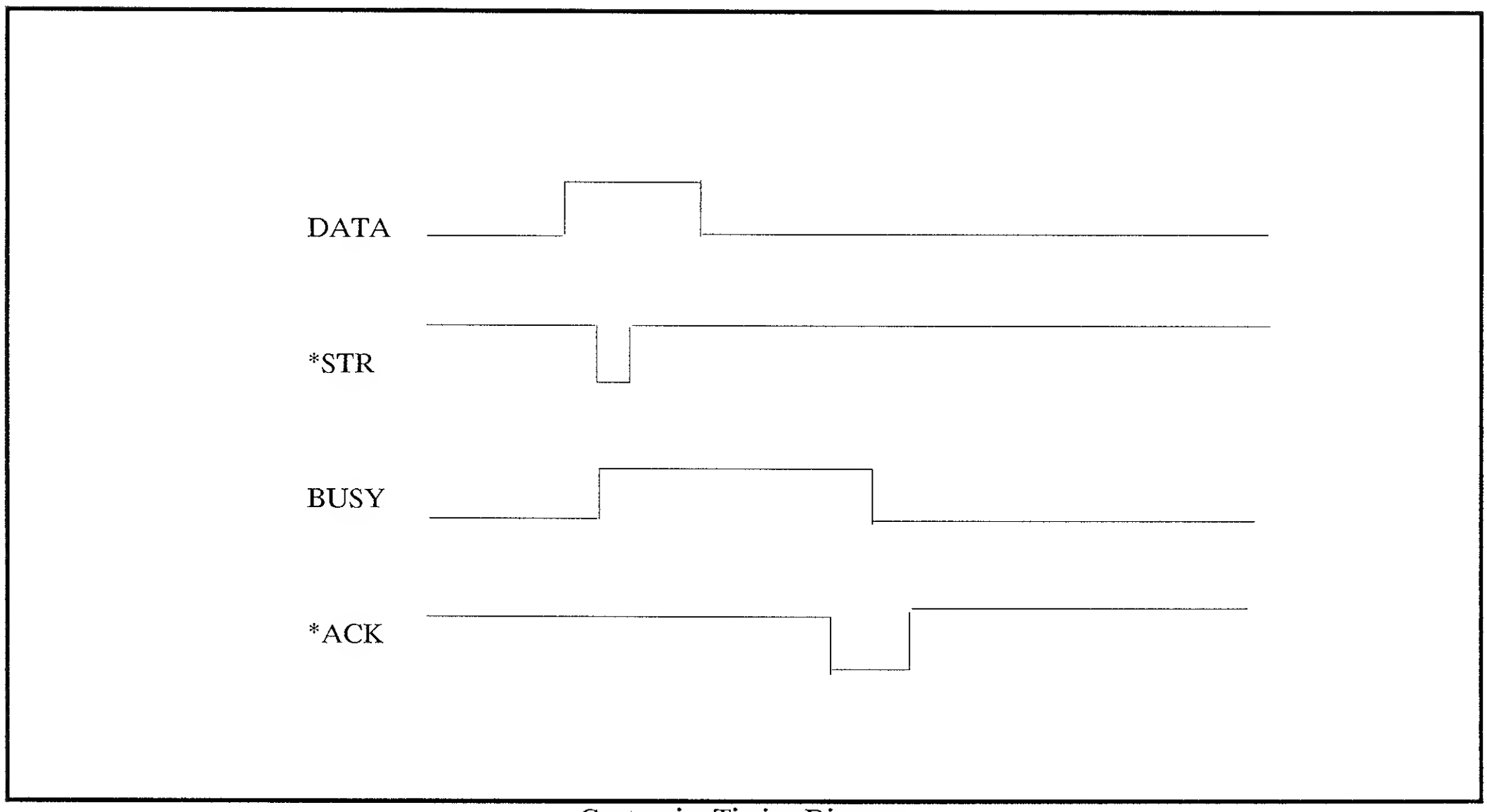
When the *STR input goes low, the interrupt occurs for the 8031. This signals the 8031 that a byte of data is available to be read at the parallel input port. The software in the interrupt service routine must first turn on the BSY (busy) output signal that goes to pin 11 on the Centronics connector. This prevents the computer sending the data from sending any more until the current byte has been processed.

Once this has been done, the system can load the proper external address in the DPTR register and execute a "MOVX a,@DPTR" instruction to read the data byte. Once read, the data byte can then be processed by the 8031. Normally, to avoid excessive delays caused by a long interrupt service routine, the 8031 will just store the byte in memory and set some sort of "processing required" flag. The true processing work required for this byte of data would then be handled by the main program.

When the byte has been read and stored, the 8031 can then turn the BSY signal to the Centronics connector "OFF" and return from the interrupt processing routine. At this point, the 8031 resumes execution of the main program where it left off.

This process is repeated each time the *STR signal from the Centronics interface is asserted.

This design is the minimum configuration for connection to any Centronics-compatible device. To achieve more efficient results, the *ACK (data acknowledge) signal may be used to signal the sending device that the current byte of data has been received and the receiving system is ready for another byte of data.



Centronics Timing Diagram

Notes:

1) At least .5 microseconds are required between the leading edge of the data being valid and the leading edge of the *STR going low.

2) A *STR pulse of at least .5 microseconds is required.

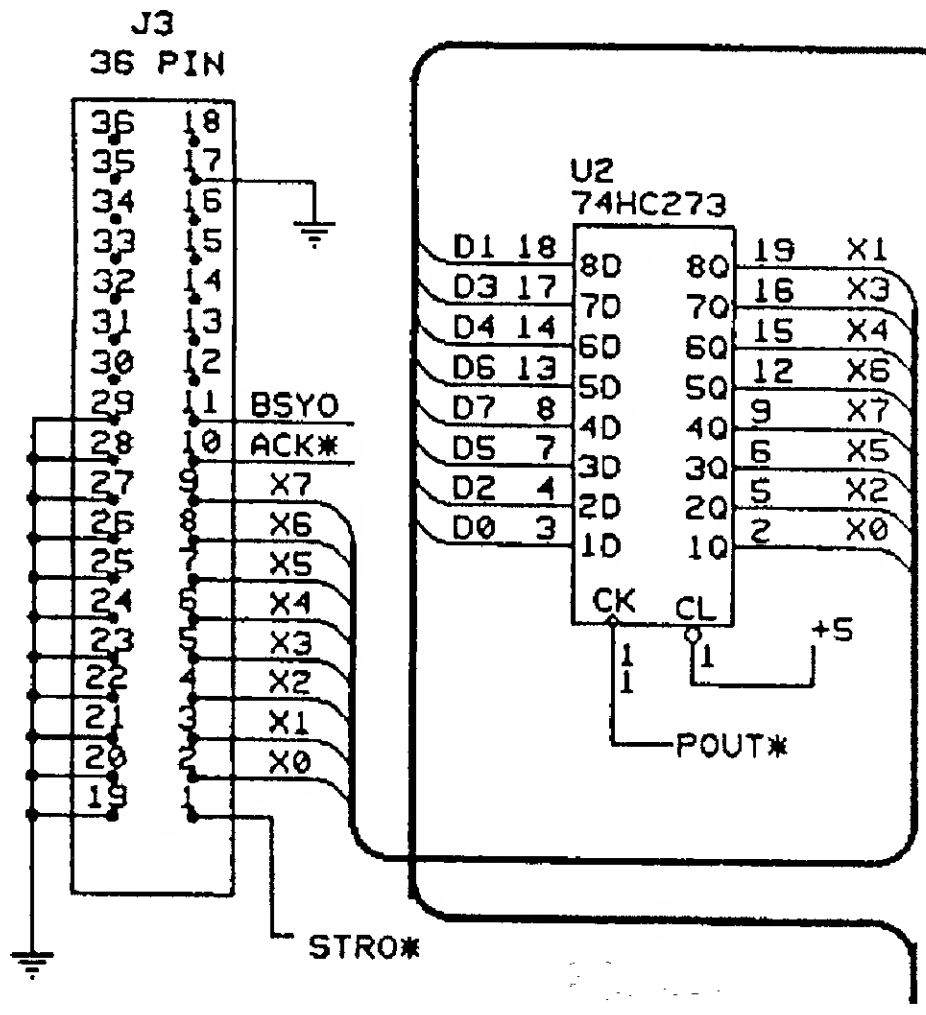
3) Data must be valid for more than .5 microseconds after the trailing edge of the *STR pulse.

```
;*************************************************
; PARALLEL INPUT CHARACTER SOFTWARE
; ASSUMES ADDRESS MAPPED TO 4010H
; INTERRUPT DRIVEN ON INT0
;     P3.1 USED FOR *ACKI SIGNAL
;     P3.0 USED FOR BUSY SIGNAL
;
;*************************************************
          .DATA

          ORG   030H

;SCRATCH PAD AREA   (30H-7FH)

INPTR:          REG   30H          ;INPUT POINTER
INPTR1:         REG   31H
OUTPTR:         REG   32H          ;OUTPUT POINTER
OUTPTR1:        REG   33H

; BEGINNING OF CODE SPACE
          .CODE

;*****************************************
;VECTOR ROUTINE TABLE
;*****************************************
          ORG   0003H
          LJMP EXINT0              ;EXT INT 0 VECTOR
                                   ;PARALLEL PORT IN


;*************************************************
; INITIALIZATION

          ORG   0040H
; SET OUTPUT STATE
          SETB C
          MOV   P3.1,C             ;ACKI/

;SET STARTUP ADR FOR IN AND OUT BUFFERS
          MOV   DPH,#80H   ; (ASSUMES ram AT 8000H)
          MOV   DPL,#00H
          MOV   INPTR,DPL
          MOV   INPTR1,DPH
;SET INTERRUPT PRIORITY'
          MOV   IP,#00000001B        ;PAR IN HI PRIORITY
;SET TIMER 0 RUNNING and INT0 to edge triggered
          MOV   TCON,#00010001B
;ENABLE INTS
          MOV   IE,#87H              ;TMR0,INT0,INT1

          RET
```

```
;*************************************************************
;VECTOR INTERRUPT ROUTINES
;*************************************************************
EXINT0          EQU  $
;SET BSYI OUTPUT ON
          SETB P3.0

          PUSH PSW                ;PARALLEL CHAR IN
          PUSH ACC
          PUSH DPL
          PUSH DPH
          MOV  A,R0
          PUSH ACC
          PUSH B
;READ CHAR FROM PAR IN
          MOV  DPTR,#4010H
          MOVX A,@DPTR
          MOV  R0,A
;LOAD BLOCK INDEX TO DPTR
          MOV  DPL,INPTR
          MOV  DPH,INPTR1
;WRITE DATA TO PROPER ADR
          MOV  A,R0
          MOVX @DPTR,A
;INCREMENT INDEX
          INC  DPTR
;STORE INDEX
          MOV  INPTR,DPL
          MOV  INPTR1,DPH


          POP  B
          POP  ACC
          MOV  R0,A
          POP  DPH
          POP  DPL
          POP  ACC
          POP  PSW

;SET BUSY OUTPUT OFF
          CLR  P3.0

          RETI
```

1.6. Centronics Parallel Output Port

A Centronics-compatible output port is useful in any design that will require direct connection to parallel printers for output. This ability adds a lot of value to the system



by providing a standard connection to hundreds of commercially available printers.

1.6.1 Hardware

The Centronics output port is the exact opposite process for the way that data is input on the input port previously discussed. The process now becomes latching the eight bits of data to be available on the output port, then to pulse the STR output signal to indicate that the byte of data is available to be read. The design has been made using a 74HC273 8-bit latch as the interface between the 8031 and the connector. The clock signal for the latch is controlled by the output of the 74HC138 3 of 8 decoder which determines the address that the data byte is written to.

The *STRO (strobe out) signal at the Centronics connector is connected to an output bit on the 8031 and is controlled by the software directly from the 8031.

The BSYO signal is also connected directly to the 8031 as an input. This signal is controlled by the receiving device.

1.6.2 Software

At a particular point in the program, the 8031 may determine that a byte (or bytes) of data need to be sent to the output port. If this is part of a print driver routine, it will occur when the system begins to send data to the printer port.

The first thing that needs to be done is to check the BUSY input from the Centronics connector. If this is simply connected to an input bit on the 8031, the bit can be read and evaluated. If it is asserted by the receiving device, the system cannot send a byte of data at this point. A software polling loop could be used to continuously check the state of this bit until it goes to a "not-busy" state.

If the port is not busy, the software can then load the proper address for the Centronics output port into DPTR. The data byte can then be loaded in the "A" register and the MOVX @DPTR,A command executed. This causes the byte of data to be latched by the 74HC273 on the output port.

Once the data has been latched, the STR output signal must now be transitioned from a HIGH state to a LOW, then back to a HIGH to indicate to the receiving computer to read the byte of data.

To maintain the timing requirements of the Centronics interface standard, a 5 microsecond delay must be inserted after latching the data before asserting the *STRO (strobe out) signal. This can be done by a simple software loop.

Next the *STRO output can be transitioned from a HIGH state to a LOW (true) state at the Centronics connector. Again, if this is connected directly to an output bit on the 8031, it is a very straight-forward matter.

A 1 microsecond delay is inserted while the *STRO signal is TRUE. The software can simply be designed to include a NOP instruction to cause a brief delay of this length.

The *STRO signal then can be transitioned from a LOW state to a HIGH state. An additional 5 microsecond delay is needed at this point to complete the strobe timing.

This completes the cycle for sending one byte on the Centronics output port. The process is repeated for each additional byte of data that is to be sent to the parallel output port.

```
;*********************************************
; CENTRONICS PARALLEL OUTPUT SOFTWARE
;
;   ASSUMES RAM BEGINNING AT ADDRESS 8000H
;*********************************************
          .DATA
          ORG  030H

;SCRATCH PAD AREA  (30H-7FH)

OUTPTR:   REG  32H        ;OUTPUT POINTER
OUTPTR1:  REG  33H

;*********************************************
;BEGINNING OF CODE
;*********************************************
          .CODE
;INITIALIZATION

;SET OUTPUTS
          SETB C
          MOV  P1.7,C          ;STR0/

;SET STARTUP ADR FOR IN AND OUT BUFFERS
          MOV  DPH,#80H
          MOV  DPL,#00H
          MOV  OUTPTR,DPL
          MOV  OUTPTR1,DPH
          RET


;*************************************
; OUPUT PARALLEL CHAR
;   ASSUMES OUTPTR+OUTPTR1  IS POINTING TO BYTE TO OUTPUT
;   ASSUMES PARALLEL OUT IS MEMORY MAPPED TO 4020H
;*************************************
PAROUT1         EQU  $

;;IF NOT BUSY...
          MOV  C,P1.5          ;EVALUATE BIT
          JC   PAROUT1
;;;SET ADR FOR READ
          MOV  DPL,OUTPTR
          MOV  DPH,OUTPTR1
;;;READ DATA
          MOVX A,@DPTR
;;;WRITE TO OUTPUT PORT
          PUSH DPL
          PUSH DPH
          MOV  DPTR,#4020H
          MOVX @DPTR,A
;;;WAIT 5 uS
          CALL FIVEUS
```

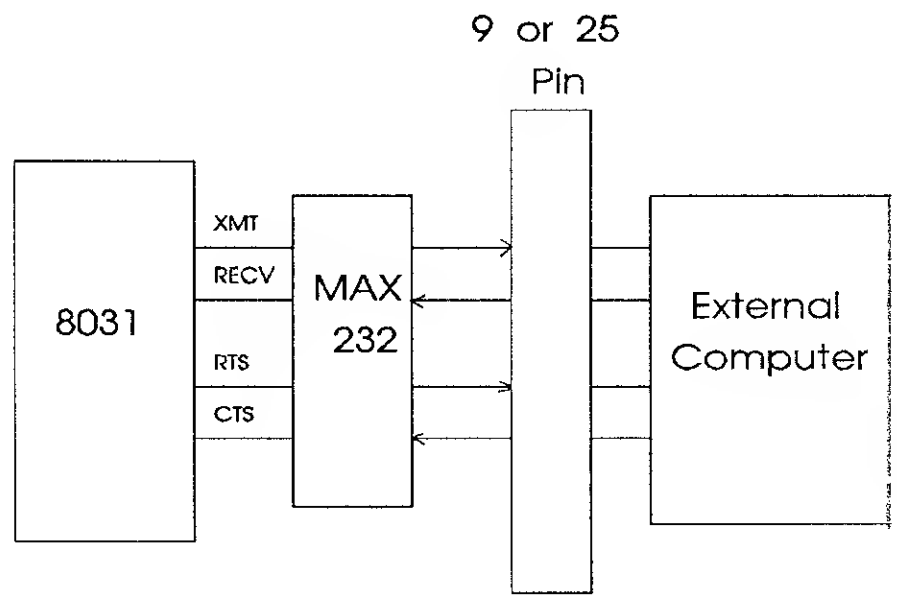
```
;;;SET STROBE OUT LOW
          CLR  P1.7
;;;WAIT 1 uS
          NOP
;;;SET STR OUT HI
          SETB P1.7
;;;WAIT 5 uS
          CALL FIVEUS
;;;INC INDEX
          POP  DPH
          POP  DPL
          INC  DPTR
;;STORE INDEX
          MOV  OUTPTR,DPL
          MOV  OUTPTR1,DPH
;ENDIF
          RET

;*****************************************
; SUBROUTINE AREA
;*****************************************

FIVEUS    EQU  $
          RET                    ;5 MICROSECOND DELAY
                                 ; AT 11.0592 XTAL
```

1.7. Interfacing to the built-in Serial Port

One of the built-in capabilities of microcontrollers in the 8051 family is the internal serial channel that can be configured for communication with other serial devices. This may be used for connection to an external terminal for providing the user interface between the outside world and the single board computer, or for any other application where serial communication may be required.

1.7.1 Hardware

The only hardware required for a RS232-compatible serial channel is a voltage conver-



sion chip and a "D" type connector to provide the proper interface to the "outside world". We have chosen to use the Maxim MAX232 chip to do the voltage conversion for this design. This chip takes the standard logic voltage levels (0V and 5V) and converts them to the voltage levels required for successful RS232 operation (in this case +10V and -10V). This chip also provides one additional input and one additional output that can be used for voltage level conversion of hardware handshaking lines. The hardware handshaking implemented in this design includes a RTS (request to send) output signal to the sending device (indicating that the 8031 is ready to receive data) and a CTS (clear to send) signal input from the external device (which the 8031 polls before sending data).

The "D" connector indicated in the sample design is the 25 pin standard. However, some systems may make use of the newer 9 pin standard that was made popular by the IBM AT computer. A sample chart showing the correct connection for the 9 pin standard is shown in Appendix B.

1.7.2 Software

The software required for operation consists of the initialization process, which configures the 8031 internally for serial communication operation. This consists of setting the Timer 1 register values to the proper settings to give the desired bit-per-second rate and configuring the interrupts to provide transmit and receive interrupt operation.

To program Timer 1 for correct operation in this design, follow the steps below:

1) Set the C/*T flag in the TMOD register to "0".

2) Program the Timer/Counter mode in TMOD to 8-bit autoreload mode.

3) Program the SCON register to Mode 1 and serial communication enabled.

4) The following table can then be used to determine the correct reload value for the timer itself.

Baud Rate	Reload Value
9600	FDh
4800	FAh
2400	F4h
1200	E8h

For the software presented in this sample design, 1200 baud is chosen for use, with the TH1 register programmed to a value of E8h and the TL1 register is programmed to 00h.

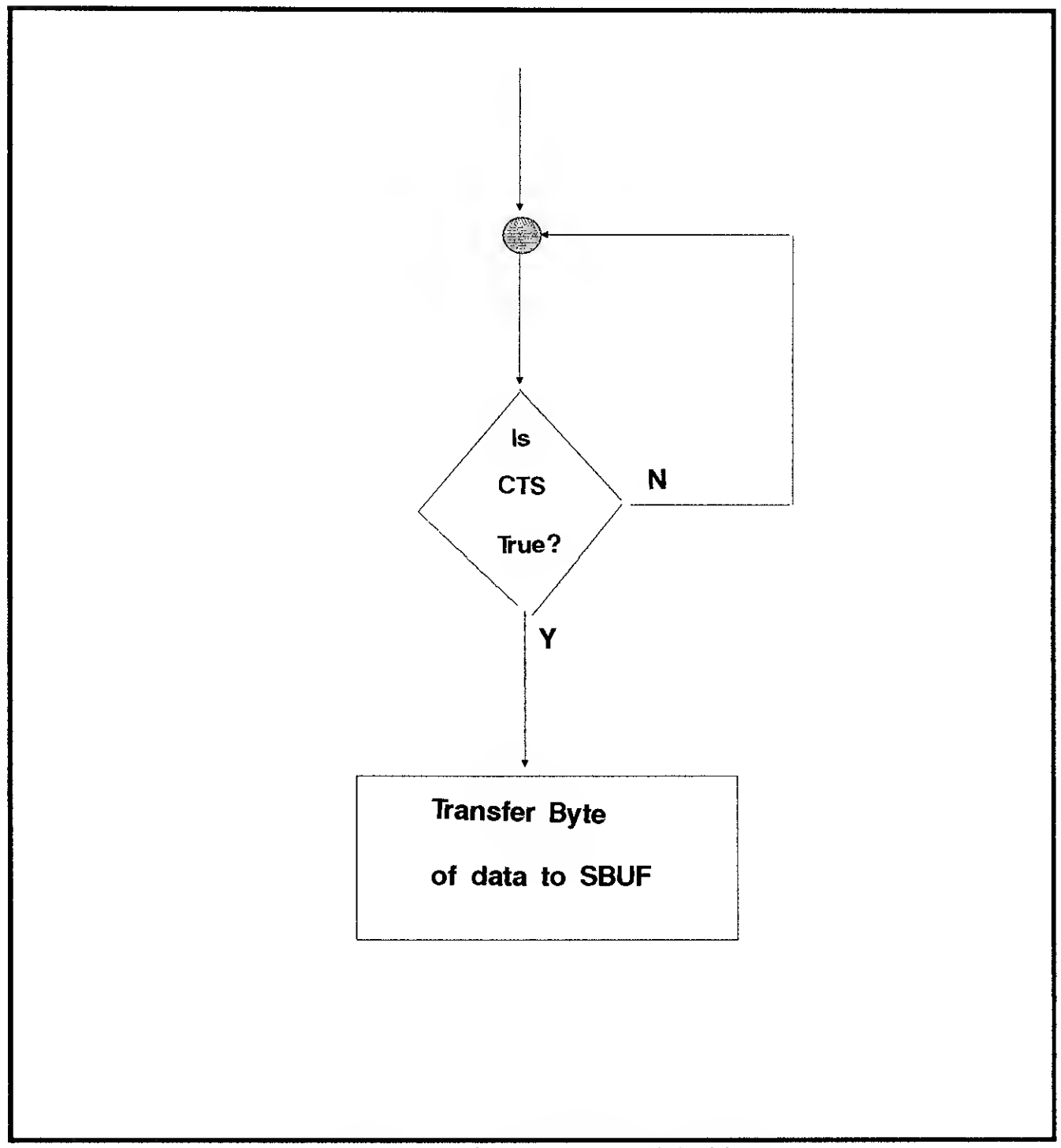
Using this mode, ten bit words are used for transmission (one start bit, 8 data bits, and one stop bit).

Once initialized, the 8031 will vector to the interrupt routine when a character has been received on the serial port or when the SBUF register is empty after transmitting a byte. One thing to watch out for when using the serial port interrupt in the 8031 is the fact that either the RI (receive interrupt) or TI (transmit interrupt) can cause the interrupt to occur. Therefore, the software must poll the RI flag to check to see if the interrupt was caused by a character being received. If not, the interrupt was caused by the transmit process signalling the 8031 that the SBUF is empty.

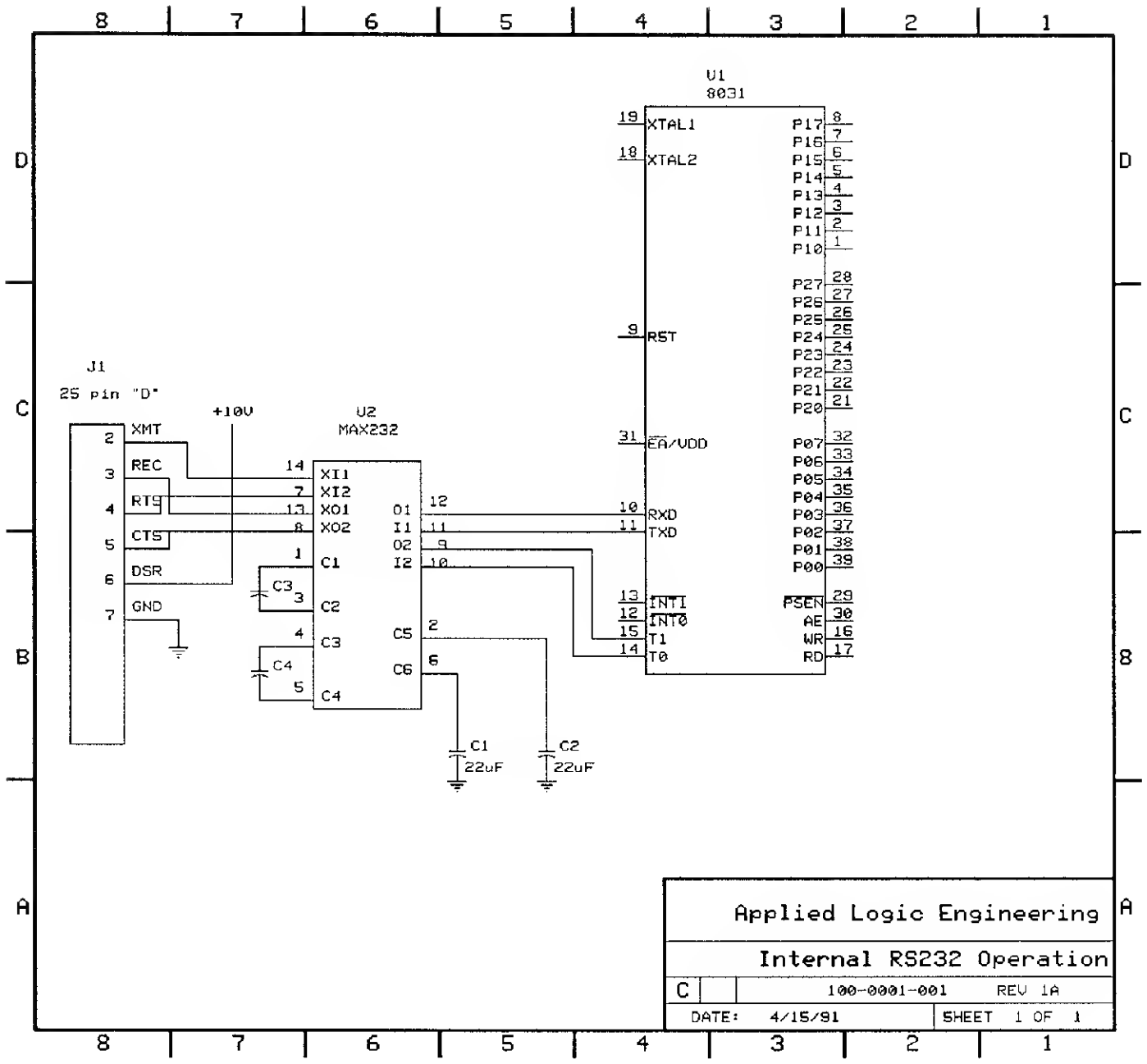
If receiving a character, the vector routine will then read the character from the SBUF register and process it before exiting the interrupt routine.

To transmit a character, the software must first look at the CTS input from the receiving

device to make sure that the receiving device is asserting this handshaking line. If this line is asserted, the 8031 can transfer the byte of data to transmit to the SBUF register. The internal UART will then take care of sending the bits out in serial format in the ten bit format described above.



Sequence Using CTS Handshaking


U1
8031
XTAL1
XTAL2
RST
EA/VDD
RXD
TXD
INT1
INT0
T1
T0
P17
P16
P15
P14
P13
P12
P11
P10
P27
P26
P25
P24
P23
P22
P21
P20
P07
P06
P05
P04
P03
P02
P01
P00
PSEN
AE
WR
RD
J1
25 pin "D"
XMT
REC
RTS
CTS
DSR
GND
+10V
U2
MAX232
XI1
XI2
XO1
XO2
O1
I1
O2
I2
C1
C2
C3
C4
C5
C6
C1
22uF
C2
22uF
Applied Logic Engineering
Internal RS232 Operation
C
100-0001-001 REV 1A
DATE: 4/15/91
SHEET 1 OF 1

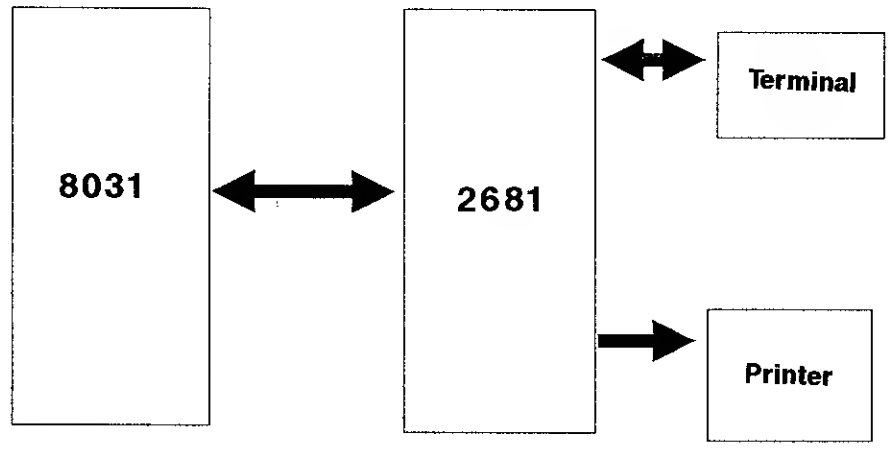
```
          .CODE
;*****************************************
; SOFTWARE FOR INTERNAL RS232 OPERATION
;         ASSUMES P3.5 IS USED FOR RTS
;                 P3.4 IS USED FOR CTS
;*****************************************

;*****************************************
; VECTOR FOR RECEIVE INTERRUPT
;*****************************************
          ORG  0023H
          LJMP SERIAL
;*****************************************
; INITIALIZATION
;*****************************************
;
;LOAD TIMER 1 VALUE FOR 1200 BAUD
          MOV  TL1,#0
          MOV  TH1,#E8H
;SET TIMER 1 TO AUTO RELOAD
          MOV  TMOD,#00100000B
;SET TIMER 1 TO MODE 1
          MOV  SCON,#50H
;SET TIMER 1 RUNNING
          MOV  TCON,#01000000B
;ASSERT REQUEST TO SEND (RTS)
          SETB P3.5
;TURN ON INTERRUPTS
          MOV  IE,#10010000B
          RET

;*****************************************
; TRANSMIT A CHARACTER
;*****************************************

; IF CTS IS ASSERTED...
XMIT2:    MOV  C.P3.4
          JNC  XMIT2
;MOVE CHAR TO SBUF REGISTER TO TRANSMIT
          MOV  SBUF,A
;WAIT FOR XMIT PROCESS TO BE COMPLETED
XMIT1:    MOV  C,TI
          JNC  XMIT1
          CLR  TI
          RET

;*****************************************
; SERVICE ROUTINE FOR INTERRUPT VECTOR-RECEIVE ONLY
;*****************************************

SERIAL:
;IF THIS IS A RECEIVE PROCESS...
          MOV  C,RI
          JNC  SERIAL1          ;NO-INT CAUSED BY XMIT
```

```
;;CLEAR RTS OUTPUT
          CLR  P3.5
;;READ CHAR
          MOV  A,SBUF
;;CLEAR RECEIVE BIT
          CLR  RI
;;REASSERT RTS
          SETB P3.5
;;RETURN FROM SERVICE ROUTINE
SERIAL1:  RETI
;ENDIF
```

1.8. Interfacing to a Dual Channel UART

If more than one serial port is required in a design, a slightly different approach can be implemented using an external DUART (Dual UART) chip, the Signetics 2681. This chip provides for two independent serial channels that can be programmed for totally independent operation.

The 2681 provides for internal baud rate generation for each channel so that no



additional chip is required to provide the oscillation rate. Also, each channel can be programmed for independent operation, meaning for example that channel A could be run at 9600 baud while channel B is simultaneously running at 1200 baud.

The advantage to this type of design is that it is relatively simple to implement and that it allows the single board computer to be interfaced to two separate serial devices, such as a terminal, modem, or serial printer.

1.8.1 Hardware

The 2681 implementation in this design is treated as an external memory device as far as the 8031 is concerned. It is connected directly to the data bus and the low four address lines (A0-A3) for internal register access. This is necessary to program the device and to send and read data from it.

The *CE (chip enable) signal for the 2681 is provided by the second 74HC138 decoder, which provides for a 16 byte block decode.

In this implementation, Serial Channel "A" is receive interrupt driven, that is to say that when characters are received on the incoming receive data line from the external device, the DUART will assert the signal on OP4 (pin 27), which is connected to an external interrupt input on the 8031. This provides for a high speed capability for receiving and transmitting from the software.

The second channel is primarily an output channel in this design for connection to a serial output device such as a printer. If data input capability is needed, status registers can be polled within the 2681 to determine character availability.

Hardware handshaking capability is provided on both channels through the use of Clear to Send/Request to Send. Unlike the single channel operation described in the previous section, the handshake lines in this example are controlled automatically by the 2681. The RTS output is asserted as long as there is buffer space available in the 2681 receive buffer. The CTS line is checked before sending a character to make sure that this signal is asserted by the receiving device. If it is not, the character will not be transmitted by the 2681.

RS232C voltage level conversions on each channel are handled by a pair of Maxim MAX232 chips that convert input voltage levels from +0V to -10V and +5V to +10V.

The circuit design around the DUART is completed by adding a simple reset circuit to the RESET input, and a 3.6864 MHZ crystal on the X1/X2 pins to provide the necessary oscillator for bit rate generation.

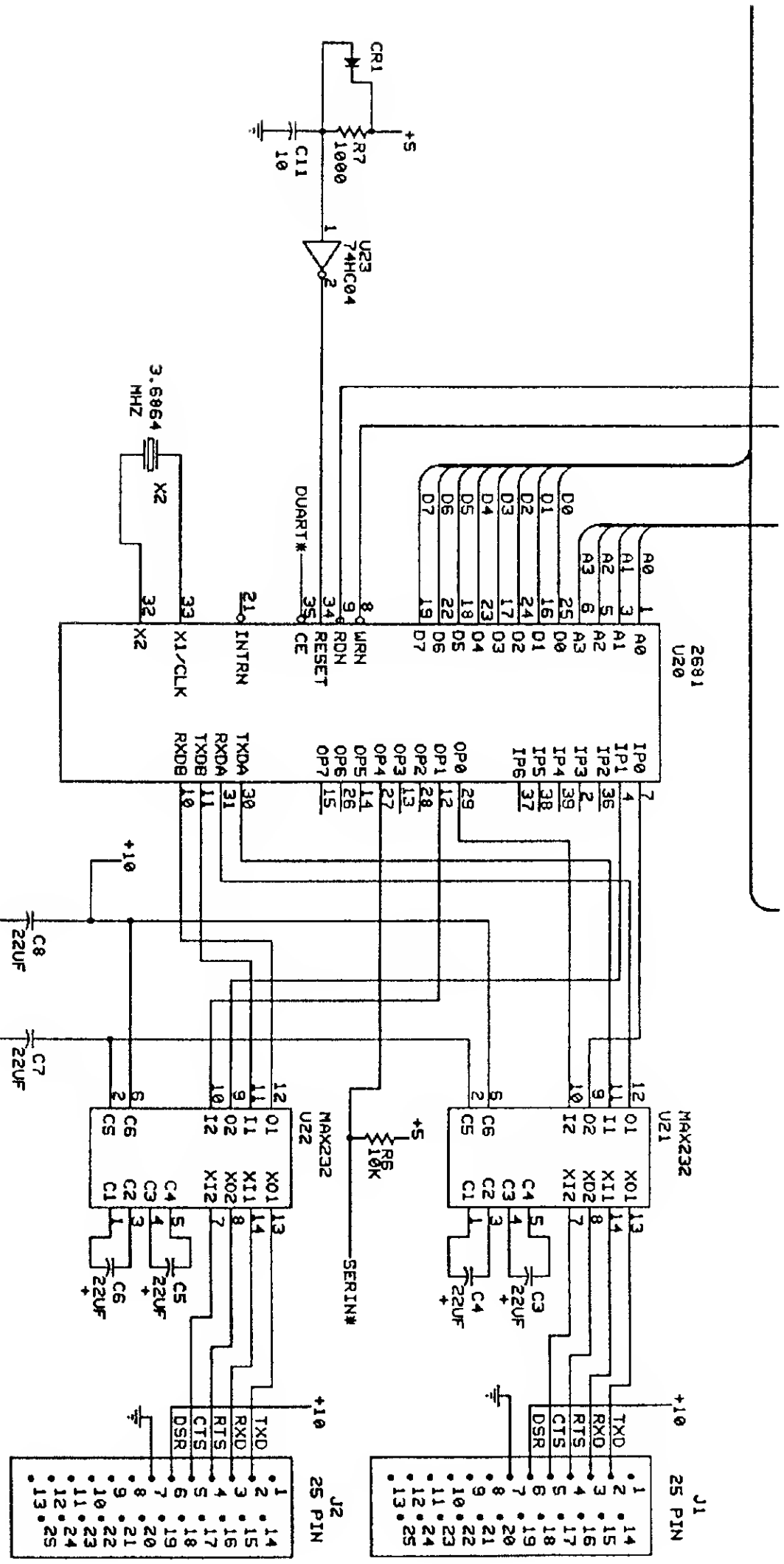
1.8.2 Software

The software for DUART control is basically done in two parts - the initialization portion and the operational portion.

The initialization part of the software sets up each individual channel of the DUART for baud rate, bits per word, parity, and number of stop bits. Interrupt operation and hardware handshaking control are also programmed in this area. For a detailed explanation of how to program this device, please refer to the Signetics data sheet on the 2681 and to the software listing that is included with this manual. The software presented in this manual programs both channels for 1200 baud, no parity, eight data bits, and one stop bit. Also, RTS and CTS hardware handshaking operation is programmed to be enabled.

The operational characteristics (i.e. baud rate, parity, etc.) for each of the serial channels can be changed "on-the-fly", but it is important to make sure that data is not coming in on the receive input while a change is being made to the device configuration.

Operational use of the DUART once it has been programmed is easy. If the channel has been configured for receive interrupt use, the 8031 will respond to the 2681 signal that a character has been received by jumping to a vector routine that would normally


CR1
R7
1000
C11
10
+5
U23
74HC04
3.6864
MHZ
X2
DUART*
2681
U20
A0
A1
A2
A3
D0
D1
D2
D3
D4
D5
D6
D7
WRN
RDN
RESET
CE
INTRN
X1/CLK
X2
IP0
IP1
IP2
IP3
IP4
IP5
IP6
OP0
OP1
OP2
OP3
OP4
OP5
OP6
OP7
TXDA
RXDA
TXDB
RXDB
+10
C8
22UF
C7
22UF
MAX232
U22
C1
C2
C3
C4
C5
C6
O1
O2
I1
I2
XO1
XI1
XO2
XI2
R6
10K
SERIN*
MAX232
U21
C1
C2
C3
C4
C5
C6
O1
O2
I1
I2
XO1
XI1
XO2
XI2
C6
22UF
C5
22UF
C4
22UF
C3
22UF
+10
TXD
RXD
RTS
CTS
DSR
J2
25 PIN
J1
25 PIN

read the character out of the device. Once read, the byte is stored in RAM for processing by the main OS software after returning from the interrupt.

Transmitting a byte on the same channel is done by polling the 2681 to make sure that is able to transmit a byte, then loading the DPTR with the correct address with the data to send in the "A" register. A MOVX @DPTR,A instruction will then send the byte of data to the device.

```
;*************************************************
; Dual Channel RS232 via Signetics 2681 DUART
; ASSUMES DUART MAPPED TO ADDRESS 4030H
;*************************************************
          .DATA
OUTPTR:   REG  30H
OUTPTR1:  REG  31H
INPTR:    REG  32H
INPTR1:   REG  33H

          .CODE

;Interrupt 1 vector

          ORG  0013H
          LJMP EXINT1          ;EXT INT 1 VECTOR
                               ;RS232 "A" RECV

;*************************************************

; Initialization

;PROGRAM DUART
          MOV  A,#10H               ;ISSUE SET POINTER CMD
          MOV  DPTR,#4032H          ;
          MOVX @DPTR,A

          MOV  DPTR,#4030H
          MOV  A,#93H                    ;MR1A
                                ;RTS-ON, NO PARITY
                                ;8 DATA BITS
          MOVX @DPTR,A
          MOV  A,#17H                    ;MR2A
                                ;CTS-ON, 1 STOP BIT
          MOVX @DPTR,A

          MOV  A,#66H                    ;1200 BAUD
          MOV  DPTR,#4031H
          MOVX @DPTR,A

          MOV  A,#05                ;RELOAD AND OUTPUT CRA
          MOV  DPTR,#4032H
          MOVX @DPTR,A


          MOV  A,#10H               ;ISSUE SET POINTER CMD
          MOV  DPTR,#403AH
          MOVX @DPTR,A

          MOV  DPTR,#4038H
          MOV  A,#93H                    ;MR1B
                               ;RTS-ON, NO PARITY
                               ;8 DATA BITS
          MOVX @DPTR,A
```

```
MOV   A,#17H                    ;MR2B
                              ;CTS-ON, 1 STOP BIT
MOVX @DPTR,A

MOV   A,#66H                    ;1200 BAUD
MOV   DPTR,#4039H
MOVX @DPTR,A

MOV   A,#05                     ;RELOAD AND OUTPUT CRB
MOV   DPTR,#403AH
MOVX @DPTR,A


MOV   A,#F0H                    ;EXT CLK/16
MOV   DPTR,#4034H               ;ACR
MOVX @DPTR,A
            MOV   A,#03         ;INTERRUPT ON - CHAN "A"
            MOV   DPTR,#4035H   ;IMR
            MOVX @DPTR,A
            MOV   A,#00H        ;SET INT TIMER VALUES
            MOV   DPTR,#4036H   ;CTUR
            MOVX @DPTR,A
            INC   DPTR          ;CTLR
            MOVX @DPTR,A
            MOV   A,#F0H        ;USE OUTPUT BITS FOR INTS
            MOV   DPTR,#403DH   ;OPCR
            MOVX @DPTR,A
            MOV   A,#0FH        ;SET H/W HANDSHAKING ON
            MOV   DPTR,#403EH   ;OUTPUT PORT
            MOVX @DPTR,A

;ENABLE INTS
            MOV   IE,#87H       ;TMR0,INT0,INT1
            RET


;*****************************************************
;OUTPUT SERIAL A CHAR
;*****************************************************
; outptr POINTS TO RAM BUFFER WHERE CHARACTER TO TRANSMIT IS

;;;IF OK TO XMIT...
            MOV   DPTR,#4035H
SAOUT5:     MOVX A,@DPTR
            ANL   A,#01H
            JZ    SAOUT5
;;;SET ADR FOR READ
            MOV   DPL,OUTPTR
            MOV   DPH,OUTPTR1
;;;READ DATA
            MOVX A,@DPTR
;;;WRITE TO OUTPUT PORT
            PUSH DPH
            PUSH DPL
```

```
            MOV   DPTR,#4033H
            MOVX  @DPTR,A
;;;INC INDEX
            POP   DPL
            POP   DPH
            INC   DPTR
;;;SAVE ADDRESS
            MOV   OUTPTR,DPL
            MOV   OUTPTR1,DPH
            RET



;*********************************************
;OUTPUT SERIAL B CHAR
;*********************************************
; outptr POINTS TO RAM BUFFER WHERE CHARACTER TO TRANSMIT IS

;;;IF OK TO XMIT...
            MOV   DPTR,#4035H
SBOUT5:     MOVX  A,@DPTR
            ANL   A,#10H
            JZ    SBOUT5
;;;SET ADR FOR READ
            MOV   DPL,OUTPTR
            MOV   DPH,OUTPTR1
;;;READ DATA
            MOVX  A,@DPTR
;;;WRITE TO OUTPUT PORT
            PUSH  DPH
            PUSH  DPL
            MOV   DPTR,#403BH
            MOVX  @DPTR,A
;;;INC INDEX
            POP   DPL
            POP   DPH
            INC   DPTR
;;;SAVE ADDRESS
            MOV   OUTPTR,DPL
            MOV   OUTPTR1,DPH
            RET



;*********************************************
;VECTOR INTERRUPT ROUTINES
;*********************************************
; INPTR POINTS TO RAM BUFFER TO STORE CHAR


EXINT1:     PUSH  PSW                 ;RS232 "A" RECV
            PUSH  ACC
            PUSH  DPL
```

```
            PUSH DPH
            MOV  A,R0
            PUSH ACC
            PUSH B

;READ CHAR
            MOV  DPTR,#4033H
            MOVX A,@DPTR
            MOV  R0,A
;LOAD BLOCK INDEX TO DPTR
            MOV  DPL,INPTR
            MOV  DPH,INPTR1
;WRITE DATA TO PROPER ADR
            MOV  A,R0
            MOVX @DPTR,A
;INCREMENT INDEX
            INC  DPTR
;STORE INDEX
            MOV  INPTR,DPL
            MOV  INPTR1,DPH

            POP  B
            POP  ACC
            MOV  R0,A
            POP  DPH
            POP  DPL
            POP  ACC
            POP  PSW
            RETI
```

1.9. Interfacing to an LCD

A common need in a single board computer design is the ability to display information that can be viewed by the system operator. Usually this type of information indicates system operation status, configuration, or any other data that would concern the system operation.

A good solution for this type of application usually is a small single or double line display that consumes very little power. A Liquid Crystal Display (LCD) can offer these and several other advantages to your design.

This manual will focus on LCDs that use the Hitachi HD44780A00 LCD controller chip (or equivalent). LCDs that use this on-board controller provide for a very simple interface to most single board computer designs. This controller contains all necessary drivers and memory capabilities to provide simple parallel data transfer in an ASCII format.

The Hitachi HD44780A00 controller is used on many different types of LCDs from various manufacturers. It controls LCDs in many different configurations, including 1 line X 8 characters, 1X16, 1X20, 1X40, 2 lines X 16 characters, 2X20, and 2X40. All configurations are software compatible, so the software drivers provided can be used in any of the above configurations.

1.9.1 Hardware

The hardware interface to the LCD consists of connecting the 8031 data bus (D0-D7) to the device, connecting address line A0 to the RS (register select) input, connecting the R/W (read/write) input to the appropriate line, and the chip enable to the proper output signal.

A single five volt supply is required for operation. Also, a potentiometer is connected to the LCD to provide variable contrast control.

This design uses a slightly different approach from other interface designs used to connect an LCD to a 8031. This design does not require any additional I/O pins on the 8031. It is designed to "look like" a standard memory device to the 8031 for both reading and writing operations.

Since the oscillator for the 8031 in the design being presented here is set at 11.059 MHZ, some conditioning on the chip enable signal coming from the 74HC138 must be made. This is due to the fact that the time between the R/*W input on the LCD and the ENABLE input when accessing the LCD must be no less than 140 ns. In order to accomplish this, the chip enable signal is "delayed" by introducing a resistor/capacitor network between the output of the 74HC138 decoder and the input of the LCD. This effectively delays the transition of the chip enable into the LCD by the required time

period.

If an oscillator less than 10 MHZ is used for the 8031, no delay is required and the chip enable can be connected directly from the memory decoding scheme to the LCD. Also, if an oscillator faster than 10 MHZ is used (other than 11.059 MHZ), you will need to determine the proper values for the resistor and capacitor in order to get the proper amount of delay time.

The *RD and *WR signals from the 8031 are combined in a single R/*W line to connect to the LCD.

1.9.2 Software

The software for interfacing to the LCD consists of routines to send command information to the device, send data to the device, read data from the device, and provide a DEVICE BUSY pause for waiting for the LCD to become ready to accept a new byte.

Because of the hardware that has been included in the design, the software can treat the LCD basically as an external memory device. A byte of data can be written to the LCD by loading the proper address in DPTR and the proper data byte in the "A" register. The "MOVX @DPTR,A" command then sends the byte to the LCD.

Once a byte has been written to the LCD, the LCD memory cannot be accessed again until it has completed the processing. The software can poll for this condition by reading the status register and testing the BUSY flag. If this flag is asserted, the LCD memory cannot be accessed.

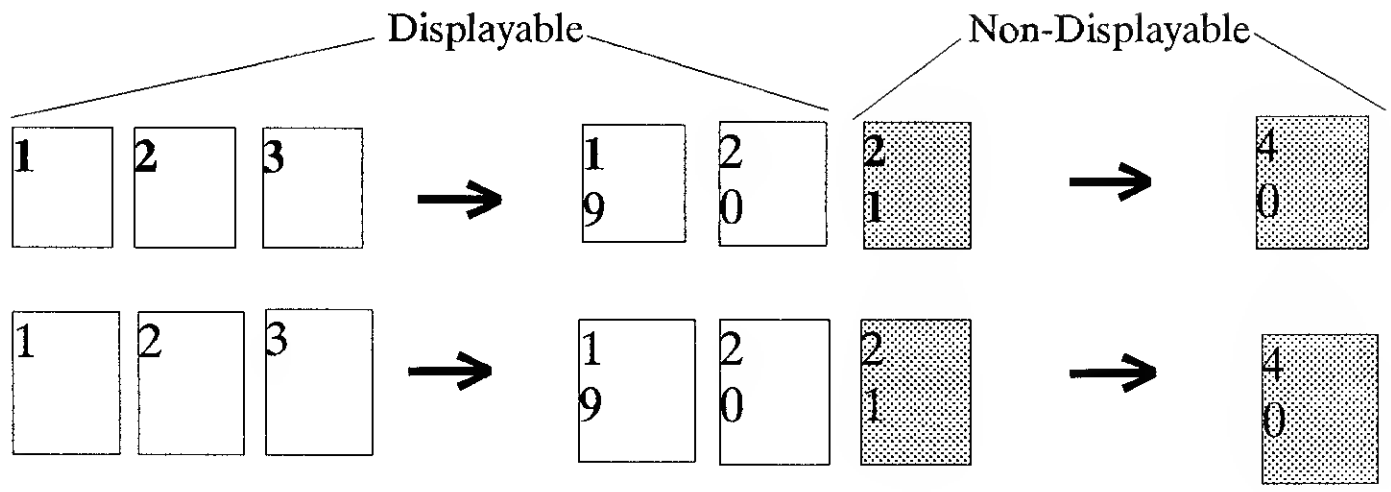
The first operation required to use the device is to program the LCD for correct operation. This consists of sending commands to the LCD to set the number of lines, number of bits per word, cursor movement direction, and turning the display on. The software provided demonstrates how this can be done.

After this has been done, characters can simply be loaded in the "A" register in the 8031 and sent to the LCD using the "MOVX @DPTR,A" command. Positioning the cursor on the LCD can be done be issuing the correct command for cursor reposition.

Software is also provided for moving strings of characters defined in a ROM table to the LCD. Also, routines to blank the LCD and position the cursor to the beginning of line two are provided.

One final set of routines provides binary to ASCII conversion and ASCII to binary conversion.

1.9.3 Different Display Configurations

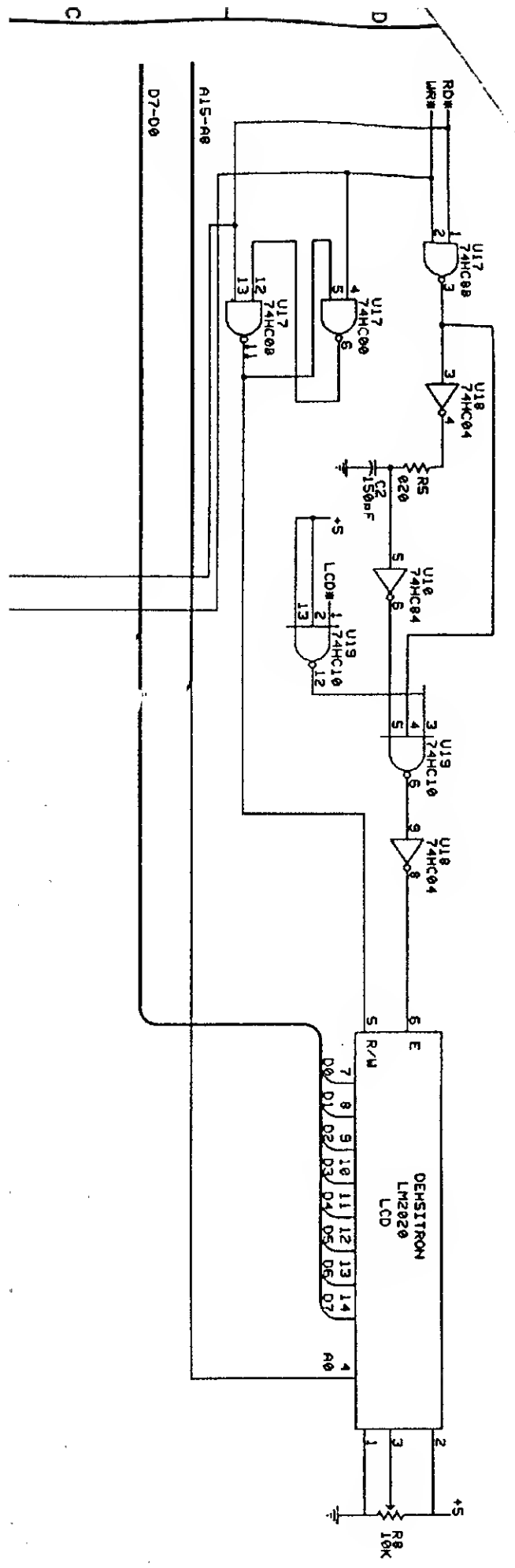


2x20 LCD - Printable Positions

The Hitachi controller for the LCD is designed to handle up to a 2 line by 40 character display unit. If you are using an LCD smaller than the 2x40, make sure that you account for the RAM positions in the LCD controller that do not correspond to character positions on the LCD.

For example, a 2x20 LCD does not have the capability to display any characters in the last 20 positions of the first or second line. The controlling software must know that these positions cannot be written to if the data is to be shown on the LCD and must either reposition the cursor to the beginning of line two or scroll the characters on line one one position to the left to create an open space for the new character at the end of the first line.

One trick that can be used if a few more bytes of RAM are required in your application program is the use of undisplayable LCD positions as general purpose RAM. Because the hardware interface in this design treats the LCD as a standard memory read/write device, any memory location in the LCD can be used for this purpose.


C
D
D7-D0
A15-A8
RD#
WR#
U17
74HC08
U17
74HC08
U17
74HC08
U18
74HC04
R5
820
C2
150pF
+5
LCD#
U19
74HC10
U10
74HC04
U19
74HC10
U18
74HC04
R/W
E
D0
D1
D2
D3
D4
D5
D6
D7
A0
OEMSITRON
LM2020
LCD
R6
10K
+5

```
;*********************************
; Software for LCD Control
;*********************************
; Note : Address mapped for LCD Instruction Register @ 4000h
                    LCD Data Register @ 4001h

          .CODE
;*********************************
; Initialization
;*********************************

          MOV  DPTR,#4000H          ;INSTR REG - LCD

;SET 8 BIT WORD-2 LINE DISPLAY-5x7 DOT FORMAT
          MOV     A,#38H
          MOVX    @DPTR,A
;WAIT FOR LCD
          CALL WAITLCD
;SET 8 BIT WORD-2 LINE DISPLAY-5X7 DOT FORMAT
          MOV     A,#38H
          MOVX    @DPTR,A
;WAIT
          CALL    WAITLCD
;SET CURSOR MOVE DIRECTION
          MOV     A,#06
          MOVX    @DPTR,A
;WAIT
          CALL    WAITLCD
;SET DISPLAY ON
          MOV     A,#0CH
          MOVX    @DPTR,A
;WAIT
          CALL    WAITLCD
;CLEAR DISPLAY
          MOV     A,#01
          MOVX    @DPTR,A
;WAIT
          CALL    WAITLCD
          RET
;****** End of Initialization


;*****************************************
; Start of Utility routines
;*****************************************
; Wait for LCD to become available

WAITLCD:  PUSH PSW
          PUSH ACC
          PUSH      DPL
          PUSH DPH

          MOV  DPTR,#4000H
```

```
WAITLCD1: MOVX A,@DPTR
          RLC  A
          JC   WAITLCD1

          POP  DPH
          POP  DPL
          POP  ACC
          POP  PSW
          RET


;*************************************
; move data from a ROM table to the LCD
; DPTR must be set to starting ROM address
; "B" register holds number of chars to transfer
; destroys contents of R4 register
;
;
TABTOLCD: MOV  R4,A
          MOVC A,@A+DPTR              ;GET CHAR

          PUSH DPH
          PUSH DPL
          MOV  DPTR,#4001H
          MOVX @DPTR,A                ;WRITE CHAR TO LCD
          CALL WAITLCD                ;WAIT FOR LCD

          POP  DPL
          POP  DPH
          MOV  A,B
          DEC  A
          MOV  B,A
          CJNE A,#00,TTL2
          RET                    ;ALL DONE

TTL2:     MOV  A,R4
          INC  A
          SJMP TABTOLCD


;*****************************************
; Blank the LCD

BLANK:    MOV  DPTR,#4000H
          MOV  A,#01H
          MOVX @DPTR,A
          CALL WAITLCD
          RET

;*************************************
; Set cursor position to Line two of the LCD
; or to the last eight chars of a 1X16 display
;
```

```
LINE2       EQU     $
            PUSH    DPL
            PUSH    DPH
            PUSH    ACC

            MOV     DPTR,#4000H
            MOV     A,#C0H
            MOVX    @DPTR,A
            CALL    WAITLCD

            POP     ACC
            POP     DPH
            POP     DPL
            RET
```

```
;*****************************************
; ASCII to Binary Conversion
; R1 points to 1st ASCII char - (R1)+1 is the second char
; Destroys contents of "B" and "R4" registers
;

ATOB:       MOV  A,@R1              ;GET 1ST CHAR

            CALL ADJ            ;CREATE BINARY NUM
            MOV  B,A
            MOV  A,#16
            MUL  AB             ;MULT *16
            MOV  B,A            ;SAVE
            INC  R1
            MOV  A,@R1              ;GET 2ND CHAR

            CALL ADJ
            ADD  A,B            ;COMBINE
            RET

ADJ:        MOV  R0,A
            ANL  A,#F0H
            CJNE A,#40H,ADJ1
            MOV  A,R0
            CLR  C
            SUBB A,#37H
            RET
ADJ1:       MOV  A,R0
            CLR  C
            SUBB A,#30H
            RET

;*****************************************
; Binary to ASCII conversion
; The "A" register holds the binary number to convert.
; "B" register is destroyed
; Uses address offset in R0 as pointer to store 2 chars
```

```
BTOA:     MOV  B,A        ;SAVE
          ANL  A,#F0H
          RR   A
          RR   A
          RR   A
          RR   A
          CJNE A,#10,BTOA1
BTOA1:    JC   BTOA2           ;A<10?
          ADD  A,#37H
          SJMP BTOA3
BTOA2:    ADD  A,#30H
BTOA3:    MOV  @R0,A
          INC  R0
          MOV  A,B
          ANL  A,#0FH
          CJNE A,#10,BTOA4
BTOA4:    JC   BTOA5
          ADD  A,#37H
          SJMP BTOA6
BTOA5:    ADD  A,#30H
BTOA6:    MOV  @R0,A
          RET
```

1.10. Bank Selection of Memory

The 8031 has the capability for directly accessing 64K bytes of program space (0000h-FFFFh with *PSEN) and an additional 64K bytes of external memory (0000h-FFFFh without *PSEN). In most single board applications, this is an adequate amount of memory.

However, in some applications, more memory space is required. A method that can be used to effectively add additional data memory to the system is called "bank selection". This scheme uses the same 64K byte external memory space addressing, but adds additional logic to expand the number of memory devices that can be selected.

1.10.1 Hardware

In the application described in this manual, the external memory space from 8000h-FFFFh will be designed to employ bank selection to expand this area from 32K bytes to 160K bytes.

Five 32K Static RAMs are used in this design to provide for 160K bytes of read/write memory.

To accomplish bank selection to these five devices, five output bits (P10-P14) on the 8031 are used to select individual 32K blocks. Address line A15 is combined with the individual block selectors with separate "NAND" gates. By setting one of the block selectors HIGH and the others LOW, one of the memory devices will be active when external data reads are made from the 8031 in the address range from 8000h-FFFFh.

1.10.2 Software

The software provided has a check for the number of SRAMS that are populated in the board in the given example. This would normally be done as part of a power up initialization so that the system would be able to know how much memory is available. A byte of data is written to the first location of each RAM and read back to verify if the RAM exists. A running count of RAMs available is kept and stored after all sites have been checked.

The software to control the bank selection process consists of one routine that is called to set the proper output select bit based on the number (0-4) that is passed to the routine

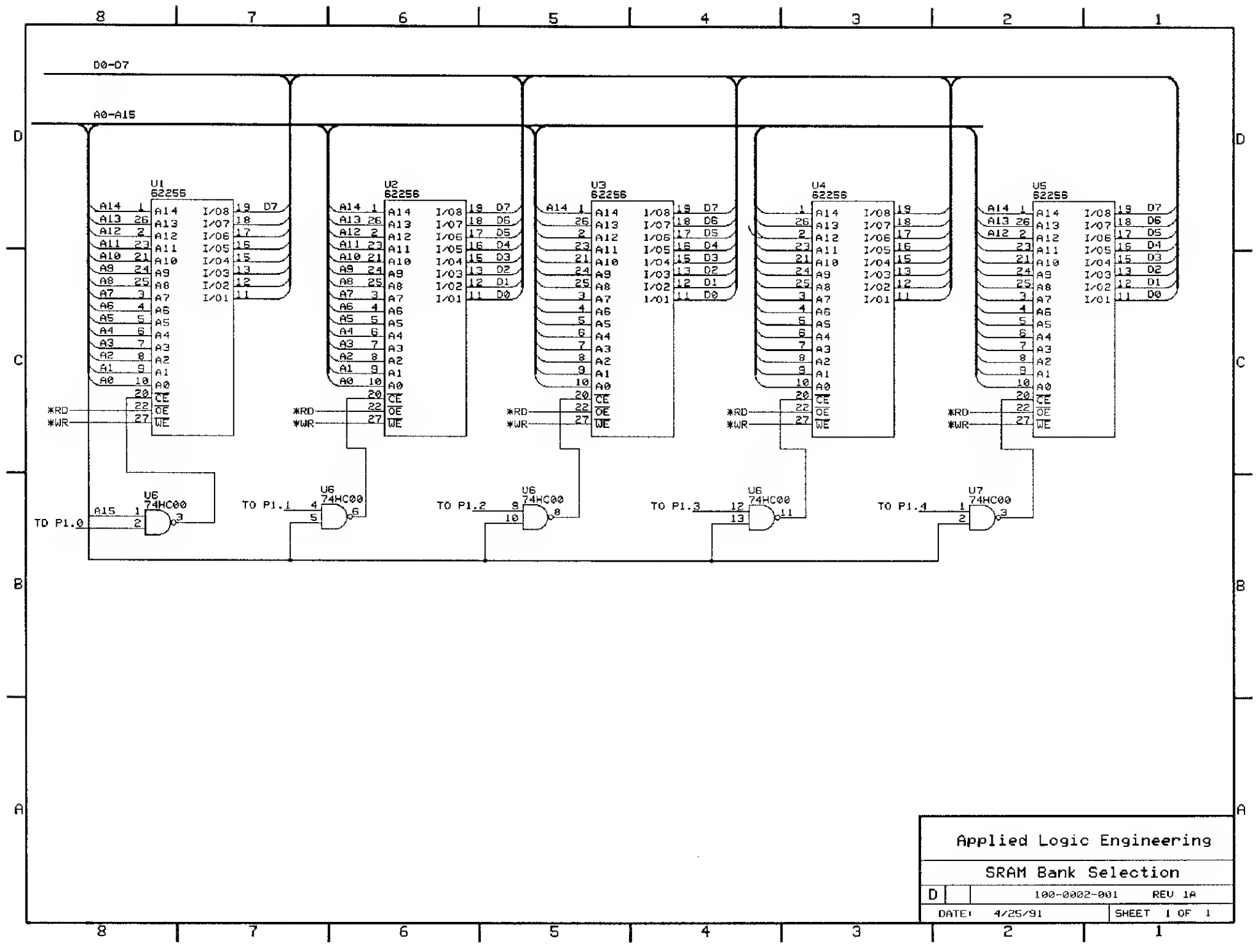
in the "A" register.

Once this routine has set the proper bank selector bit, the user is free to access external memory from 8000h-FFFFh and can expect that the proper memory device will be selected.

Additional software may be required if the bank selected memory is to be treated as "consecutive" memory, that is to say that the user wants to one device to be automatically selected after writing to the last byte (FFFFh) in the previous device.

An example of this is if the software has BANK 1 selected (32K). If all 32K bytes are written to in consecutive order and additional memory is required, software could be written that would automatically switch the bank selector output that is active from BANK 1 to BANK 2 when the last byte of BANK1 (FFFFh) is written.

This can provide a "virtual" 160K byte memory block that can be accessed by the main program.


D0-D7
A0-A15
U1
62256
U2
62256
U3
62256
U4
62256
U5
62256
*RD
*WR
U6
74HC00
U7
74HC00
A15
TO P1.0
TO P1.1
TO P1.2
TO P1.3
TO P1.4
Applied Logic Engineering
SRAM Bank Selection
100-0002-001
REV 1A
DATE: 4/25/91
SHEET 1 OF 1

```
;*************************************************
; SOFTWARE FOR BANK SELECTION OF 5 32K SRAM CHIPS.
; USES 8031 PINS P1.0-P1.4 FOR CONTROL OF SRAM TO USE IN
; EXTERNAL MEMORY FROM 8000H-FFFFH.
;*************************************************

          .DATA
;INTERNAL REGISTER DECLARATIONS (any available reg can be
used)

RAMMAX:   REG  36H        ;NUMBER OF SRAMS IN BOARD

;*************************************
          .CODE

; INITIALIZATION

;CLR ALL OUTPUTS TO START
          CLR  C
          MOV  P1.0,C          ;SRAM SEL
          MOV  P1.1,C
          MOV  P1.2,C
          MOV  P1.3,C
          MOV  P1.4,C

;CHECK FOR NUMBER OF SRAMS (1-5) POPULATED ON BOARD
          MOV  A,#00H
CHKRAM:   CALL BLKSEL
          MOV  DPTR,#8000H
          MOV  B,A        ;SAVE COUNT
          MOV  A,#A7H          ;CHK BYTE VALUE
          MOVX @DPTR,A
          NOP
          MOVX A,@DPTR         ;READ BACK
          CJNE A,#A7H,CHKRAM1
          MOV  A,B
          INC  A          ;INC TO NEXT SRAM SITE
          CJNE A,#05,CHKRAM    ;JUMP BACK IF NOT DONE
          MOV  B,#05           ;ALL BLKS OK
CHKRAM1:  MOV  A,B
          MOV  RAMMAX,A  ;SAVE MAX BLK COUNT (1-5)

;SETUP STARTUP BLOCK SELECTOR TO FIRST RAM SITE
          MOV  A,#00H
          CALL      BLKSEL
          RET



;*************************************************
; SUBROUTINE AREA
;*************************************************
```

```
;*************************************************
;BLKSEL - BLOCK SELECTER
;         SELECTS A RAM SITE FOR USE
;         ENTER WITH SITE NUMBER (0-4) IN A REGISTER
;*************************************************
BLKSEL          EQU  $

                CLR  P1.0
                CLR  P1.1
                CLR  P1.2
                CLR  P1.3
                CLR  P1.4

                CJNE A,#00H,BSEL1
                SETB P1.0
                SJMP BSEL5
BSEL1:          CJNE A,#01H,BSEL2
                SETB P1.1
                SJMP BSEL5
BSEL2:          CJNE A,#02H,BSEL3
                SETB P1.2
                SJMP BSEL5
BSEL3:          CJNE A,#03H,BSEL4
                SETB P1.3
                SJMP BSEL5
BSEL4:          CJNE A,#04H,BSEL5
                SETB P1.4

BSEL5:          RET
```

1.11. Appendix A - List of Vendors

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

8 bit Microcontroller Handbook P/N 270645-002

- covers the 8051 family of microcontrollers

Densitron
2540 West 237th Street
Torrance, CA 90505
(213) 530-3530

- manufacturer of LCDs.

Signetics
811 E. Arques Avenue
Sunnyvale, CA 94088
(408) 991-2000

data sheet on SCN2681 DUART

Maxim Corporation
120 San Gabriel Drive
Sunnyvale, CA 94086
(408) 737-7600

data sheet on MAX232 RS232 driver/receiver chip.

0.1. Appendix B: Connection to an External Computer

When deciding on whether to use a parallel or serial connection between the single board computer and an external computer, several items must be considered. The parallel connection usually provides for only a one way transfer of data from the external computer to the single board computer. If communication back to the external computer is needed, a bidirectional parallel port would be required to provide a communication path back to the computer.

An alternative to a bidirectional parallel port is a serial communication port. A standard RS232 port is generally available on most PCs, while a bidirectional serial port is standard on most newer PCs, but not necessarily included on older PCs.

A serial port can provide for a better communication scheme, but its disadvantage can be the speed at which data can be transferred. Normally, on most PCs, the transfer rate may be limited to 9600 bits per second.

0.1. RS-232 Serial Connection

After successfully completing the necessary logic and voltage conversions on the single board computer for the serial port, the next step is connecting the serial port on this board to an external source. Normally, this is some sort of external computer (PC, Mac, or other) that could be used for a number of purposes.

An external computer connected to the single board computer may be required for providing a user of the system access to the single board computer. This may be necessary to program various parameters, to retrieve accumulated data from the single board computer, or to get current status information from the process that the single board computer is controlling.

0.1.1 RS232-C Connector Pinouts

Connectors used for serial RS232 purposes on computers generally conform to a standard pinout. The twenty-five pin sub "D" connector was chosen several years ago as the normal connector used for this purpose. However, in the past couple of years, the nine pin sub "D" connector that became popular with the release of the IBM AT personal computer has gained popularity. Both connectors will be described in this section.

The signals present on these connectors are defined as follows:

25 Pin	Signal Description	9 pin
1	Chassis Ground	-
2	Transmitted Data	3
3	Received Data	2
4	Request To Send	7
5	Clear to Send	8
6	Data Set Ready	6
7	Signal Ground	5
8	Carrier Detect	1
20	Data Terminal Ready	4

Transmitted Data, Received Data, and Signal Ground are the only necessary signals for connection between devices. The remaining signals defined above are used for hardware handshaking or for modem control signals. For the purposes of this manual, a single hardware handshaking system using RTS and CTS will be described. There are other combinations and uses of the handshaking lines, but this method has been proven to work on most hardware.

The signals defined above have equivelent meaning whether they are used on the 25 pin connector or the 9 pin connector. The signals required for connection from a single board computer to an external PC will be described in the following section.

0.1.2 Connection to an External Computer

The pinouts above are used in equipment that are defined under the EIA RS-232C standard as "Data Terminal Equipment" (or DTE). Data Terminal equipment is normally defined as computers or terminals that are the primary source of data transmission. To make the connection from the 25 or 9 pin connector on the single board computer to an external PC, an understanding of what the various signals on the RS232 standard mean will be required.

The most important pins on the connector are the ones that allow the data to be transferred between the devices. The pin labeled TRANSMITTED DATA is the output pin while transmitting data, and the pin labeled RECEIVED DATA is the input pin while receiving data. Because of this, a connection is made from the TRANSMITTED DATA pin on the single board computer to the RECEIVED DATA pin on the external computer. Conversely, the TRANSMITTED DATA pin on the external computer is connected to the RECEIVED DATA pin on the single board computer. This provides for data transmission paths in both directions

SIGNAL GROUND is the only other required signal that must be connected between single board computer and the external computer. This provides the electrical reference for the TRANSMITTED and RECEIVED data signals.

If hardware handshaking is to be implemented between the two devices, the lines REQUEST TO SEND and CLEAR TO SEND on each device must be connected in a similar manner to the TRANSMITTED DATA and RECEIVED DATA signals. The REQUEST TO SEND (RTS) signal from the single board computer must be connected to the CLEAR TO SEND (CTS) signal on the external computer. Similarly, the RTS on the external computer must be connected to the CTS signal on the single board computer.

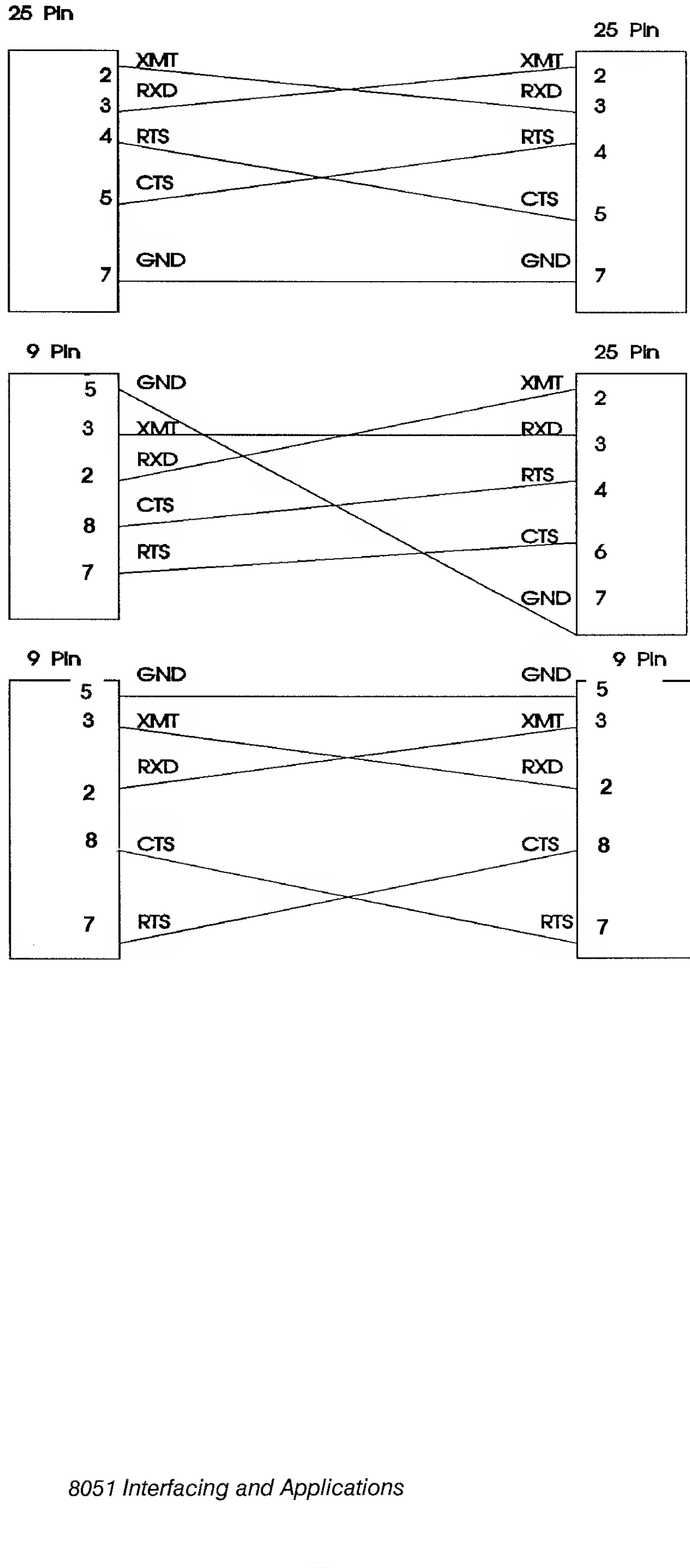
Software operation is discussed in the manual sections that cover the 8051 operation as it pertains to serial communication. Basically, it consists of polling the CTS input signal before data is transmitted, and controlling the RTS output signal to signal the transmitting device to stop sending data to the receiving device.

0.1.3 Cabling

The above connection scheme comprises a standard configuration known as a "null modem" cable that can be used to directly connect two DTE devices together. Because both of the devices are configured as DTE, they cannot be cabled together using a "one-for-one" cable. This type of cable, if used, would cause outputs to be connected to outputs and inputs connected to inputs. To avoid this, the signals must be crossed in the cable to conform to the proper wiring configuration.

In addition to making sure that the proper signals are connected, you must also be aware of the defintion in pinouts between the 25 pin connector and the 9 pin connector. The same signals are used in both connector pinouts, but the signals appear on different pins. If your design uses a 25 pin connector, for example, and the external computer uses a 9 pin connector, a cable will need to be constructed that makes the proper connection for this connector combination.

In schematic form, the possible cable connections would look the the following:



Other possible RS232 connection options

The above explanation of a standard null modem cable has been proven to work in most systems. However, the external computer that you are connecting the single board computer to may require a slightly different configuration. The next section discusses possible problems that may occur and some remedies to provide for proper operation.

Problem : External Computer does not transmit when connected.
Possible solutions ;
Some computer serial boards require that the DSR (Data Set Ready) input on its connector be at a TRUE state before sending data. This can be accomplished by tying the DSR input to the CTS input so that the single board computer will drive both inputs TRUE when it is ready to receive data. Alternatively, the DSR input can be tied to a voltage source of between +10V and +15V to hold the signal at a TRUE state at all times.

If this is attempted and the external computer will still not send data, another possible problem may be the CD (Carrier Detect) input on the computer's serial port. Some system boards and/or software may require that this input also be at a TRUE state before it will allow data to be transmitted. The CD input can be tied to a voltage source of between +10V and +15V to hold the signal at a TRUE state at all times.

Problem: Data is lost during transmission.
Possible solutions :
If data is sent from one source (either the external computer or the single board computer) and is received incomplete or garbled, make sure that the transmitter is not sending data at a rate that is too fast for the receiving device.

If a high baud rate is being used, handshaking between devices may be required in order for the receiving devive to control the data flow so that characters coming in from the transmitter will not be lost before the receiver gets a chance to read and process them. Use the RTS/CTS connection described above to implement hardware handshaking control.

Another alternative is the use of software handshaking via the XON/XOFF protocal. In this scenario, handshaking between devices is done by sending one byte codes from the receiving device to the transmitting device to control the flow of data.

When the receiving device cannot accept any data from the transmitting device, it sends an XOFF (ASCII 13h) to the transmitting device. Upon receiving this code, the transmitting device goes into a software loop without sending data. Once the receiving device has finished processing the data it has received and is ready for additional data to be sent, it sends an XON (ASCII 11h) to the transmitting device. When the transmitting device receives this byte, it resumes sending information on the serial channel.

This type of data flow control is very useful when communicating between devices that cannot use hardware handshaking, as in communication between modems over a phone line. Since data bytes are used, the need for hardware connection between RTS/CTS is eliminated.

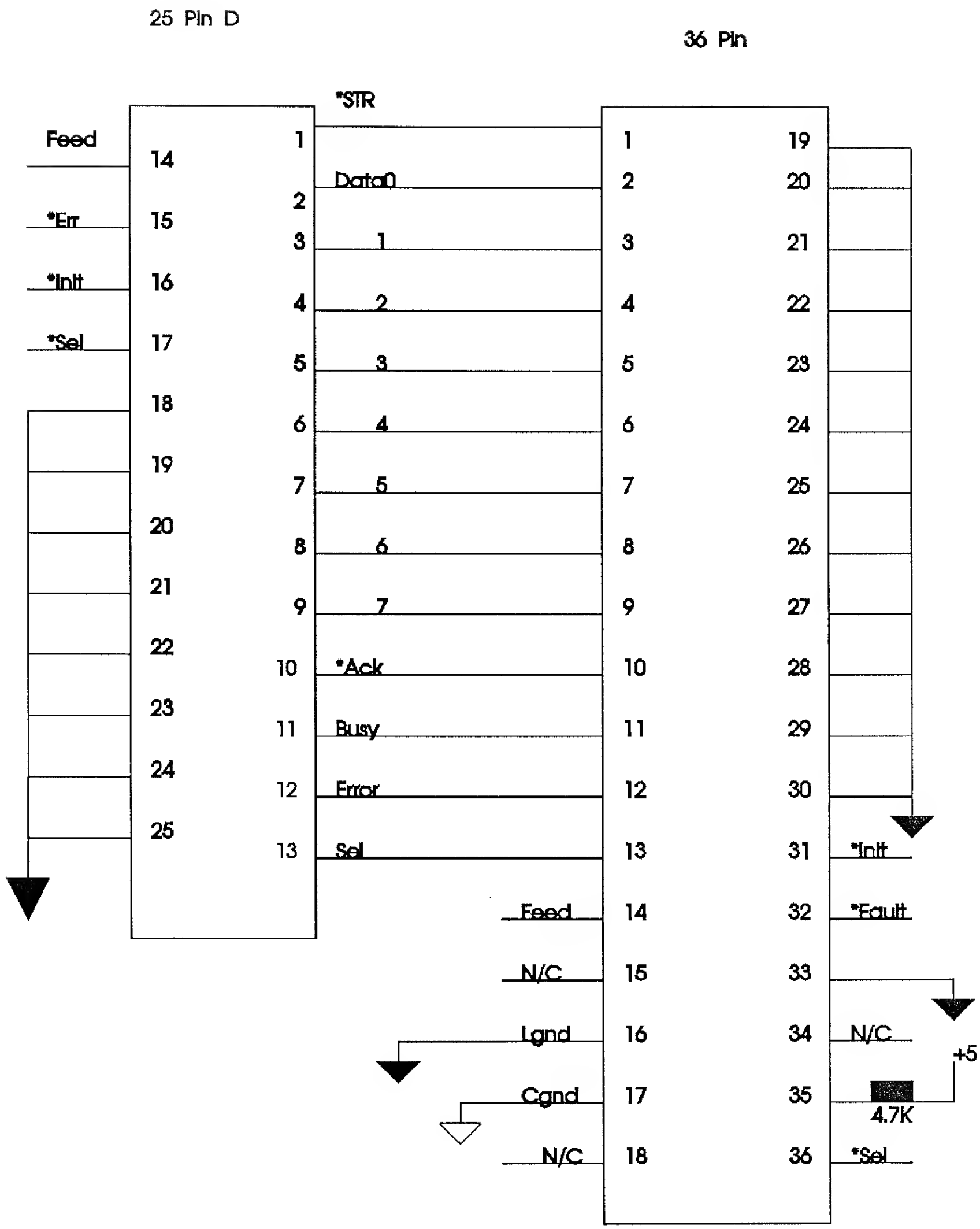
0.2. Centronics Interface Cabling

If a higher speed, one-way transmission is preferred in your system design, the parallel data communication scheme defined by the Centronics standard provides a good method for data transfer.

The interface cabling for connection of the parallel Centronics interface uses thirty six pin conductors that carry the signals for the data transfer and the control signals .

In an external computer that provides for a standard Centronics connector, a "one for one" cable can be used to connect to the single board computer that is described in this manual.

More commonly, the design may require that you connect a PC-type computer to the single board design. The IBM PC standard for parallel connection uses a 25 pin sub "D" connector rather than the standard 36 pin connector. All signals are present on the 25 pin connector that are required for the parallel interface.



IBM PC 25 pin to Centronics 36 Pin Cable