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

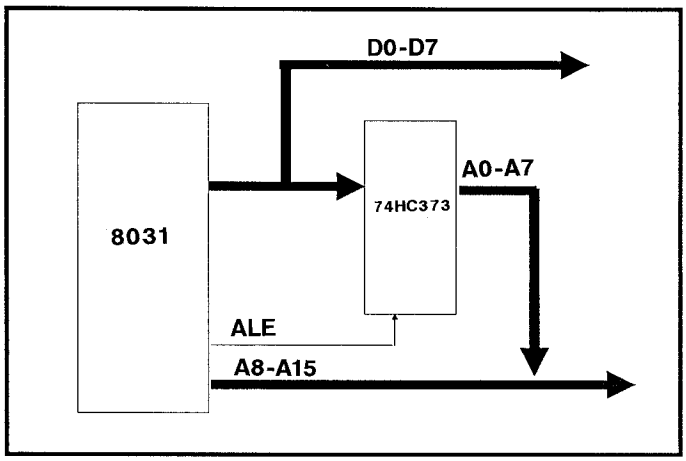
Incorporating an 8031 into a single board computer design is relatively straight forward. The chip can be configured in many different ways, but for the purpose of this discussion, various design decisions have been made and will be discussed in detail. For a detailed discussion on the capabilities of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

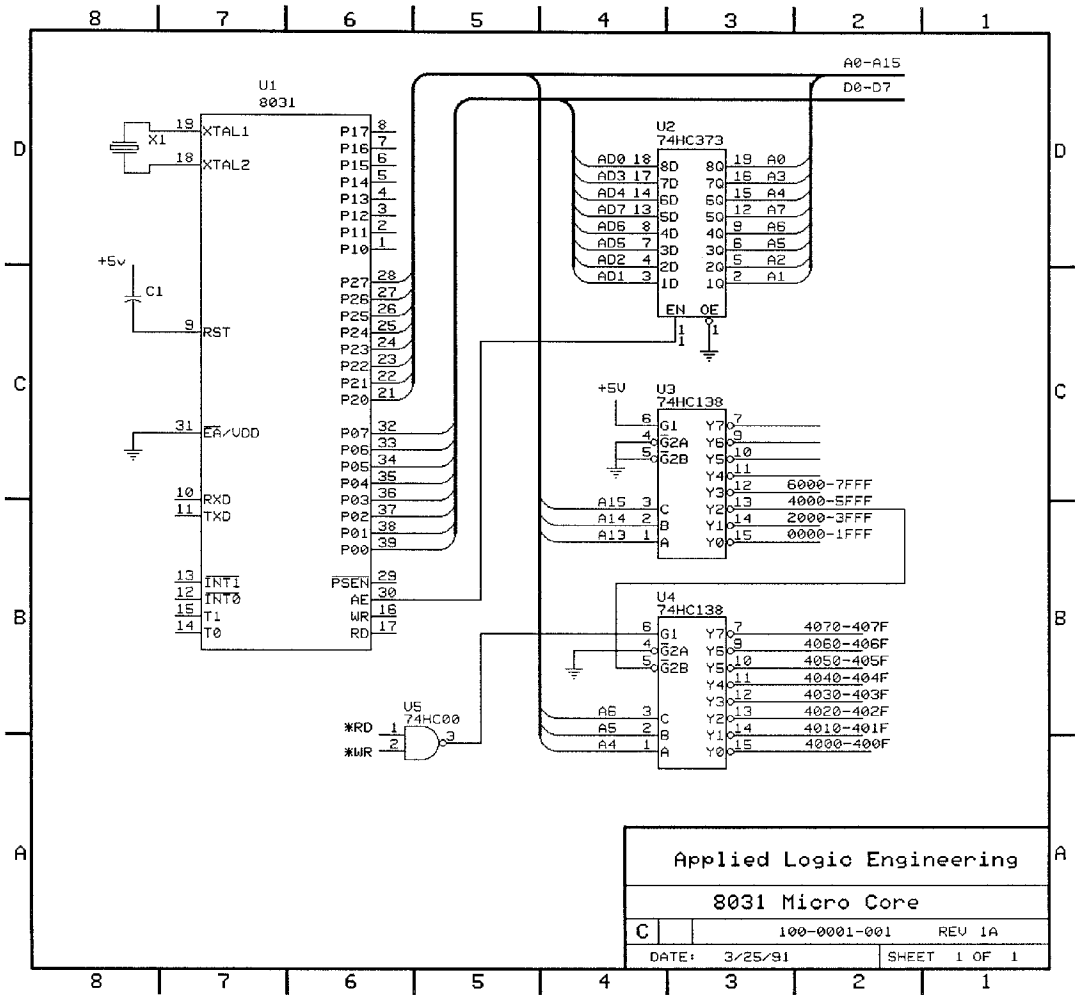
EPROM), this pin must be tied HIGH to allow the 8751 to use the internal ROM for program space.

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
X1
XTAL1
XTAL2
+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
A0 A3 A4 A7 A6 A5 A2 A1
EN OE
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 requirements.

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

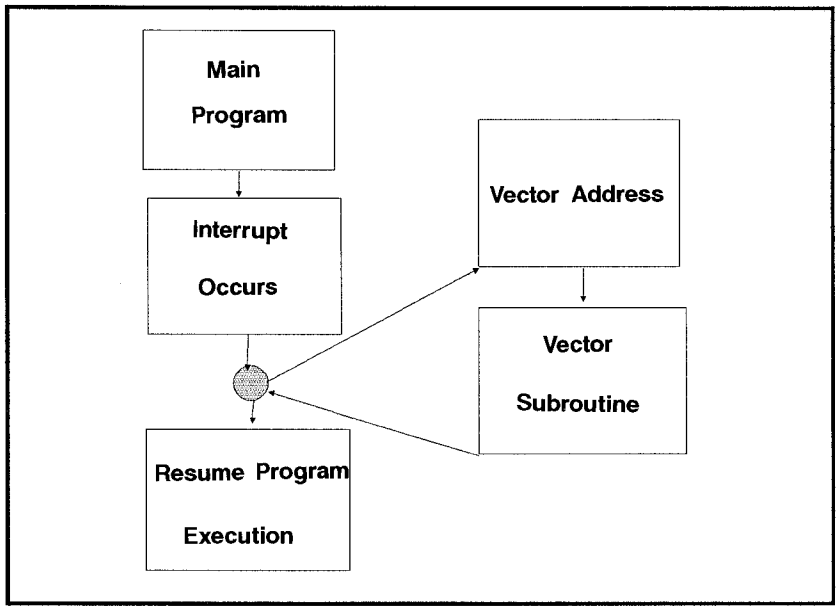
the output table would then be:

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.

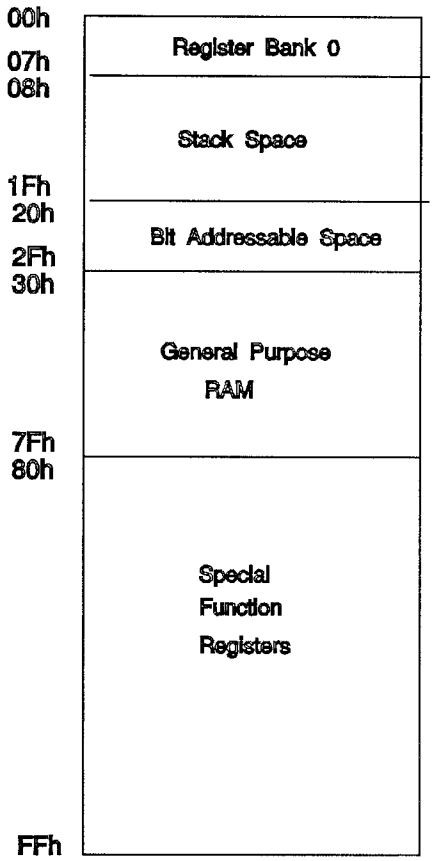
2) Also at power up, the stack pointer is initialized to 07h and incremented to 08h. This default condition assumes register bank 0 will be used for the R0-R7 registers. Also, banks 1-3 for these registers will not be available because this area will be used as the stack grows during use. If all of this 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
```