RROS

(ROM-RESIDENT OPERATING SYSTEM)

User's Manual

RIGEL CORPORATION
P.O. Box 90040, Gainesville, Florida
(352) 373-4629, FAX (352) 373-1786
www.rigelcorp.com, tech@rigelcorp.com

The abbreviation PC used throughout this guide refers to the IBM Personal Computer or its compatibles. IBM PC is a trademark of International Business Machines, Inc. MS Windows is a trademark of Microsoft, Inc.

Information in this document is provided solely to enable use of Rigel products. Rigel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Rigel products except as provide in Rigel's Customer Agreement for such products.

Rigel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Rigel retains the right to make changes to these specifications at any time without notice. Contact Rigel Corporation or your Distributor to obtain the latest specifications before placing your order.

Copyright © 1987-2002 by Rigel Corporation.

Rigel Corporation's Software License Agreement

This Software License Agreement ("Agreement") covers all software products copyrighted to Rigel Corporation, including but not limited to: Reads51, rLib51, RbHost, RitaBrowser, FLASH, rChpSim, Reads166, and rFLI.

This Agreement is between an individual user or a single entity and Rigel Corporation. It applies to all Rigel Corporation software products. These Products ("Products") includes computer software and associated electronic media or documentation "online" or otherwise.

Our software, help files, examples, and related text files may be used without fee by students, faculty and staff of academic institutions and by individuals for non-commercial use. For distribution rights and all other users, including corporate use, please contact:
Rigel Corporation, PO Box 90040, Gainesville, FL 32607
or e-mail tech@rigelcorp.com

Terms and Conditions of the Agreement

1. These Products are protected by copyright laws, intellectual property laws, and international treaties. Rigel Corporation owns the title, copyright, and all other intellectual property rights in these Products. We grant you a personal, non-transferable, and non-exclusive license to use the Products. These Products are not transferred to you, given away to you or sold to you.

   Non-commercial use: These Products are licensed to you free of charge.

   Commercial use: You must contact Rigel Corporation to find out if a licensing fee applies before using these Products.

2. You may install and use an unlimited number of copies of these Products.

3. You may store copies of these Products on a storage device or a network for your own use.

4. You may not reproduce and distribute these Products to other parties by electronic means or over computer or communication networks. You may not transfer these Products to a third party. You may not rent, lease, or lend these Products.

5. You may not modify, disassemble, reverse engineer, or translate these Products.

6. These Products are provided by Rigel Corporation "as is" with all faults.

7. In no event shall Rigel Corporation be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the Product, even if Rigel Corporation has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitations of consequential or incidental damages, the above limitations may not apply to you.

8. Rigel Corporation makes no claims as to the applicability or suitability of these Products to your particular use, application, or implementation.

9. Rigel Corporation reserves all rights not expressly granted to you in this Agreement.

10 If you do not abide by or violate the terms and conditions of this Agreement, without prejudice to any other rights, Rigel Corporation may cancel this Agreement. If Rigel Corporation cancels this Agreement; you must remove and destroy all copies of these Products.

11. If you acquired this Product in the United States of America, this Agreement is governed by the laws of the Great State of Florida. If this Product was acquired outside the United States of America all pertinent international treaties apply.

HARDWARE WARRANTY

Limited Warranty. Rigel Corporation warrants, for a period of sixty (60) days from your receipt, that READS software, RROS, hardware assembled boards and hardware unassembled components shall be free of substantial errors or defects in material and workmanship which will materially interfere with the proper operation of the items purchased. If you believe such an error or defect exists, please call Rigel Corporation at (352) 373-4629 to see whether such error or defect may be corrected, prior to returning items to Rigel Corporation. Rigel Corporation will repair or replace, at its sole discretion, any defective items, at no cost to you, and the foregoing shall constitute your sole and exclusive remedy in the event of any defects in material or workmanship. Although Rigel Corporation warranty covers 60 days, Rigel shall not be responsible for malfunctions due to customer errors, this includes but is not limited to, errors in connecting the board to power or external circuitry. This warranty does not apply to products which have been subject to misuse (including static discharge), neglect, accident or modification, or which have been soldered or altered during assembly and are not capable of being tested.

DO NOT USE PRODUCTS SOLD BY RIGEL CORPORATION AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS!

Products sold by Rigel Corporation are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

THE LIMITED WARRANTIES SET FORTH HEREIN ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

YOU ASSUME ALL RISKS AND LIABILITY FROM OPERATION OF ITEMS PURCHASED AND RIGEL CORPORATION SHALL IN NO EVENT BE LIABLE FOR DAMAGES CAUSED BY USE OR PERFORMANCE, FOR LOSS PROFITS, PERSONAL INJURY OR FOR ANY OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES. RIGEL CORPORATION'S LIABILITY SHALL NOT EXCEED THE COST OF REPAIR OR REPLACEMENT OF DEFECTIVE ITEMS.

IF THE FOREGOING LIMITATIONS ON LIABILITY ARE UNACCEPTABLE TO YOU, YOU SHOULD RETURN ALL ITEMS PURCHASED TO RIGEL CORPORATION PRIOR TO USE.

Return Policy. This policy applies only when product purchased directly from Rigel Corporation. If you are not satisfied with the items purchased, **prior to usage**, you may return them to Rigel Corporation within thirty (30) days of your receipt of same and receive a full refund from Rigel Corporation. This does not apply to books. Books are non-returnable.

Please call (352) 373-4629 to receive an RMA (Returned Merchandise Authorization) number prior to returning product. You will be responsible for shipping costs. All returns must be made within 30 days of date of invoice and be accompanied by the original invoice number and a brief explanation of the reason for the return.

Return merchandise in original packaging.

All returned products are subject to a $15 restocking charge. "Custom Items" are not returnable.

Repair Policy. If you encounter problems with your board or software after the 60 day warranty period, please call Rigel Corporation at (352) 373-4629 or email tech@rigelcorp.com for advice and instruction.

Rigel Corporation will test and attempt to repair any board. You will be responsible for shipping costs and repair fees. If you send a detailed report of the problems you encountered while operating the board, Rigel Corporation will inspect and test your board to determine what the problem is free of charge. Rigel Corporation will then contact you with an estimated repair bill. You will have the choice of having the board fixed, returned to you as is, or purchasing a new board at a reduced price. Rigel Corporation charges repair fees based on an hourly rate of $50.00. Any parts that need to be replaced will be charged as separate items. Although Rigel Corporation will test and repair any board, it shall not be responsible for malfunctions due to customer errors, this includes but is not limited to, errors in connecting the board to power or external circuitry.

Board Kit. If you are purchasing a board kit, you are assumed to have the skill and knowledge necessary to properly assemble same. Please inspect all components and review accompanying instructions. If instructions are unclear, please return the kit unassembled for a full refund or, if you prefer, Rigel Corporation will send you an assembled and tested board and bill you the price difference. You shall be responsible for shipping costs. The foregoing shall apply only where the kit is unassembled. In the event the kit is partially assembled, a refund will not be available, however, Rigel Corporation can, upon request, complete assembly for a fee based on an hourly rate of $50.00. Although Rigel Corporation will replace any defective parts, it shall not be responsible for malfunctions due to errors in assembly. If you encounter problems with assembly, please call Rigel Corporation at (352) 373-4629 for advice and instruction. In the event a problem cannot be resolved by telephone, Rigel Corporation will perform repair work, upon request, at the foregoing rate of $50.00 per hour.

Governing Law. This agreement and all rights of the respective parties shall be governed by the laws of the State of Florida.

Table of Content

1 THE ROM-RESIDENT OPERATING SYSTEM

RROS manages the prototyping board and cooperates with Reads. RROS has a command processor which can be accessed by an ASCII terminal or by the PC running a terminal emulator. The Reads is an intelligent interface with the board, which has hot keys that invoke several RROS commands to accomplish higher level tasks. Many of the RROS routines are available as user-accessible system calls.
The ROM-resident firmware consists of 6 major components:

1. An initialization routine
2. A command processor
3. User-accessible system calls
4. System variables
5. Interrupt Vectors
6. Debug utilities

Each of these components is explained below.

2 The Initialization Routine

This is a short routine that is invoked at power up or when the reset button on the board is pressed. The following actions are taken:

disable interrupts
set stack pointer to 4Fh (stack will start at 50h)
initialize hardware:
    select register bank 0
    set the interrupt vector table at FF00h
    initialize system software flags
    initialize the serial port to run at 9600 Baud with parameters 8 bits, no parity, and 1 stop bit

3 The Command Processor

The system will branch to the command processor if no auto-exec routine is present. The monitor commands are grouped under 12 single-letter commands. One or more of these commands are issued by Reads while interacting with the board. These commands may also be given by an ASCII terminal. The monitor commands are grouped according to their function and listed below.

Function	**Monitor Commands**
Read/Modify Data,	X, C, D, R
Read/Modify Special Function Registers	P
Load/Execute Program	L, G
Debug	B, K
Miscellaneous	H

The list of monitor commands is displayed with the **H** command while the monitor program is in effect. The **H** command displays the following table.

```
B xxxx          sets Break point at address xxxx
C xxxx-xxxx     displays Code memory
D xx-xx         displays internal Data ram
```

```
D xx=nn          modifies internal Data ram
D xx-xx=nn       fills a block of internal Data ram
G xxxx           Go - starts executing at address xxxx
H                Help - displays monitor commands
K                Kills (removes) break point
L                down Loads Intel hex file into memory
P x              displays data on Port x
P x=nn           modifies data on Port x to nn
R                displays the contents of the Registers
S                displays Special function register addresses
S xx-xx          displays Special function registers
S xx=nn          modifies Special function registers
S xx-xx=nn       fills Special function registers
X xxxx-xxxx      displays eXternal memory
X xxxx=nn        modifies eXternal memory
X xxxx-xxxx=nn   fills eXternal memory
```

A single-letter command may be followed by up to 3 parameters. The parameters must be entered as hexadecimal numbers. Each 'x' above represents a hexadecimal digit (characters 0..9, A..F). Intermediate spaces are ignored. Alphabetic characters are converted to upper case. The length of the command string must be 16 characters or less. The command syntax is:

```
Letter [address][-address][=data]<CR>.
```

For example, the monitor command

```
X 92C1
```

will display the contents of external memory location 92C1h. The command

```
X 92C1 - 92CF
```

will display the contents of consecutive memory locations from 92C1h to 92CFh. Similarly, the command

```
X 92C1-92CF=3F
```

will modify the contents of the memory locations from 92C1h to 92CFh, inclusively, to 3Fh. The contents of these memory locations may be verified to be 3Fh by the command

```
X92C1-92CF
```

The **C** command is identical to the X command except that code memory is displayed, not external data memory. Also, in the MCS-51 architecture, writing to code memory is not allowed. If code and external data memory banks are overlapping, then code memory can effectively be modified by the X command. Overlapping external data and code memory banks is the default architecture of the development boards. The C command is only useful if the code and external memory banks are jumper selected to be non-overlapping.

The **D** command is similar to the X command. It displays or modifies internal RAM memory. The 8031 contains 128 internal RAM locations and the 8032, 256 internal RAM locations. Notice that, in this case, the memory addresses are limited to 2 hexadecimal digits.

The **P** command allows viewing or modifying the current state of the processor ports. Viewing the state of a port is equivalent to reading the port as an input port. Modifying the port contents outputs a byte to the port. The 8031 and 8032 have 4 ports. Notice that ports 0 and 2 are used by the processor for memory address and data busses. In addition, 4 bits of Port 3 are used by the system. Bits of Port 3, P3.0 and P3.1 are used

by the serial port as the receive data and transmit data lines. P3.6 and P3.7 are used in accessing memory as the write and read control lines. Modifying bits P3.0 and P3.1 may affect the current data being transferred between the host and the board. Application programs should not write to ports 0 or 2, or bits P3.6 or P3.7 of Port 3.

The **G** and **L** commands are used to down load a program into RAM and run this program. The L command puts the board into a receive mode. The program should then be sent to the board in the Intel Hex format. Once downloading is complete, the program may be run by the G command. The parameter that follows the G command is the starting point of the program. Notice that several programs may be loaded into RAM, each one run by a G command followed by its starting address. The details of the down load-and-run process are hidden from the user when Reads is used.

The **R** command displays the contents of the 4 register banks and the accumulator (a), the b register (b), the program status word (psw), the data pointer (dptr), the stack pointer (sp), and the program counter (pc).

The commands **B** and **K** are used in debugging. Their use is described in the following sections. Again, the details of their use are hidden from the user when Reads is used.

The **H** command displays the help screen, summarizing the available monitor commands.

4 General Purpose Routines (System Calls)

The ROM-resident firmware contains many general purpose subroutines that can be called by user-written programs. Some of these subroutines are used by the system in carrying out the monitor functions. The system calls are classified by their function below.

Function	Subroutine Names
Serial Communication	chkbrk
	beep
	cret
	crlf
	getbyt
	getchr
	getchrx
	inkey
	print
	prthex
	prsphx
	prtstr
	sndchr
System	break
	delay
	init
	mdelay
	os_return
	sdelay
	setintvec
Miscellaneous	ascbin
	binasc
	display
	percent

Access to these routines is provided through a jump table located in low ROM memory. The application program can call these routines by name if the following header of equate pseudo-ops is included in the application program.

```
; ------------------------------------------------------
; system calls              registers used
; ------------------------------------------------------
ascbin              equ   0100h ;     a, r2, error flag
autoexec            equ   0103h
beep                equ   0106h ;     none
binasc              equ   0109h ;     a
break               equ   010Ch ;     a, (reads accumulator)
chkbrk              equ   010Fh ;     a, (reads serial port)
cret                equ   0112h ;     a
crlf                equ   0115h ;     a
delay               equ   0118h ;     a
display             equ   011Bh ;     a
getbyt              equ   011Eh ;     a, b
getchr              equ   0121h ;     a
getchrx             equ   0124h ;     a
init                equ   0127h
inkey               equ   012Ah ;     a
mdelay              equ   012Dh ;     a
os_return           equ   0130h
percent             equ   0133h ;     a
print               equ   0136h ;     a, dptr
prsphx              equ   0139h ;     a, r2
prtstr              equ   013Ch ;     a
prthex              equ   013Fh ;     a, r2
sdelay              equ   0142h ;     a
setintvec           equ   0145h ;     a, dptr
sndchr              equ   0148h ;     a
```

Then, the application program simply calls, say, subroutine getchr as follows.

```
        .
        .
      lcall getchr
        .
        .
```

The registers used by these routines appear in the header as comments. If the application program uses these registers, the registers should be pushed before the system call. The source code of these general purpose routines is available from our web site www.rigelcorp.com.

A short description of user-accessible system calls is given below. The source code for the system calls is in the file syscalls.src, and in the appendix at the end of this document.

ascbin - assumes that the contents of the accumulator is a hexadecimal digit, that is in the interval '0'..'9', 'A'..'F', or 'a'..'f'. Provided that the accumulator holds a valid hexadecimal character, it is converted to binary and returned as the lower nibble of the accumulator. The high nibble is cleared to 0000b. If the accumulator does not hold a valid hexadecimal character, the system error flag (errorf) is set.

autoexec - checks the RROS system variable if a routine is to be executed after power up. The presence of a start up routine is indicated by the routines start address (other than 0000h or FFFFh) placed after location

480h. If an autoexec routine exists the message "hit any key to abort" is sent out the serial port. If no characters are received from the serial port, a long jump performed to the start up routine. If no such routine is present or if a character is received through the serial port within 2 seconds of the abort message, control is turned over to the RROS command processor. The program status word (PSW) may be affected.

binasc - converts the low nibble of the accumulator to a hexadecimal character in the range '0'..'9' or 'A'..'F'. The high nibble is ignored. The program status word (PSW) may be affected.

beep - sends a bell character, ASCII 7, to the host through the serial port. No registers are affected.

break - compares the contents of the accumulator to the break character, ASCII 3 (Ctrl-C). If the accumulator holds the break character, control is passed back to the RROS command processor. The program status word (PSW) may be affected.

chkbrk - reads a character from the serial port and compares it to the break character, ASCII 3. If equal, control is passed back to the monitor command processor. The program status word (PSW) may be affected.

cret - outputs a carriage return, ASCII 0Dh, through the serial port. The accumulator (a) is affected.

crlf - outputs a carriage return, ASCII 0Dh, followed by a line feed character, ASCII 0Ah through the serial port. The accumulator (a) is affected.

delay - executes a dummy loop to suspend the execution nn milliseconds where nn is the contents of the accumulator.

display - converts the low nibble of the accumulator to the corresponding seven-segment-display pattern. Upon return, the accumulator holds the 7 bits, acc.0 corresponding to segment a, and acc.6, segment g. The bits are cleared if the corresponding segment is to be lit. Thus, the pattern will drive the common anode seven-segment-displays on the RMB-S.

getbyt - waits to receive 2 characters (2 bytes) from the serial port. If the characters are valid hexadecimal digits (0..9, A..F), then the ASCII-represented byte is converted to binary and placed in the accumulator (a). The accumulator (a) and the b register (b) are affected.

getchr - **getchrx** wait for a character to be received through the serial port. The character is then returned in the accumulator (a). Routine getchr clears the most significant bit of the character (byte), whereas getchrx returns all 8 bits.

init - invokes the initialization routine which initializes the interrupt vector table, sets the stack to 4fh, clears software flags, and sets the serial communications to 9600 Baud, 8 data bits, 1 stop bit and no parity bits. It affects the accumulator, r0, and dptr.

inkey - peeks at the serial port to see if a character has been received. If so, the character is returned in the accumulator (a). If not, a null (0) is returned in the accumulator (a). The accumulator (a) is affected.

mdelay and **sdelay** - execute dummy loops for timing purposes. The period from when mdelay is called to its return is exactly 1 millisecond. Routine sdelay takes exactly one second from its call to its return. These delay routines are exact only when a 12Mhz system clock is used, and when there are no interrupt routines in the background.

os_return - turns control over to the monitor. Since the monitor resets the stack (to 4fh), either a call or a jump instruction may be used to branch to the monitor.

percent - converts the binary fraction in the accumulator to the binary coded decimal (BCD) fraction in the interval [0..99]. For example, the binary value 80h is converted to 50 BCD. The BCD value is returned in the accumulator. This routine uses a look up table for speedy execution rather than computing the BCD value.

print and **prtstr** - send a string of characters out through the serial port. Print and prtstr use the accumulator (a) and the data pointer (dptr). The string to be sent can be of arbitrary length, provided that it terminates with the null character (0). Prtstr sends a null-terminated string pointed to by dptr. Prtstr is useful if a string in a table of strings (such as error messages) is to be printed.

The routine print assumes that the string immediately follows the call to print. It is appropriate when the message is embedded in code. An example is given below.

```
   .
   .
lcall print
db    "Hello Everybody", 0Dh, 0Ah, 0
   .
   .
```

The define byte (db) pseudo-op allocates 18 bytes the code memory immediately following the long call instruction. The first 15 bytes are filled with the ASCII codes of the letters of 'Hello Everybody'. The following 3 bytes contain the ASCII codes for carriage return (0Dh), line feed (0Ah), and the null character (0) which indicates the end of the string. The carriage return and line feed will start a new line after the string 'Hello Everybody' has been displayed.

prthex and **prsphx** - convert the binary value of the accumulator into 2 hexadecimal digits. Prsphx and prthex then send these two ASCII digits, high digit first, out through the serial port. Before termination, prsphx sends a space (ASCII 20h) out through the serial port. These routine use the accumulator (a) and register R2 during the binary to hexadecimal conversion.

setintvec - modifies the interrupt vector table so that interrupt source, from 0 to 11, indicated by the value of the accumulator (a), is directed to the interrupt service routine whose starting address is held in the data pointer (dptr). See Section 5.6 for a detailed description of the interrupt vector table. Except for the two registers (a) and (dptr), and the program status word (psw), setintvec does not affect any registers. The 14 interrupt sources of the 80537 are listed below. The other microcontrollers have a subset of these interrupt sources.

source number		description
0	int0	external interrupt 0
1	tint0	timer 0 overflow interrupt
2	int1	external interrupt 1
3	tint1	timer 1 overflow interrupt
4	sint	serial port interrupt
5	exint	timer 2 overflow interrupt
6	adcint	analog-to-digital converter
7	int2	external interrupt 2
8	int3	external interrupt 3
9	int4	external interrupt 4
10	int5	external interrupt 5
11	int6	external interrupt 6
12	s1int	auxiliary serial port interrupt
13	ctfint	compare timer overflow interrupt

Notice that the source numbers follow the default priorities of the interrupts (see the microcontroller manufacturer's data book for more information).

As an example, let t0isr be the interrupt service routine for the timer 0 overflow interrupt in an application program. The interrupt is directed to t0isr by the following instructions.

```
        mov   a, #1       ; select interrupt source 1

        mov  dptr, #t0isr ; address of the service routine

        lcall setintvec
          .
          .
          .
          .
t0isr:                    ; the interrupt service routine starts here
          .
          .
          .               ; various application-specific
          .               ; instructions
          .

       reti               ; the interrupt service routine ends here
```

sndchr - sends the contents of the accumulator out through the serial port. This routine waits until the serial transmit operation has been completed. The accumulator (a) is affected.

5 System Variables

The ROM-resident firmware uses several internal registers for the system. All of the monitor commands use register bank 0. The stack is initialized to 4Fh, so that the first byte pushed is placed in internal location 50h. Stack does not grow beyond 16 bytes (50h..5Fh) when the monitor functions or the host mode debugging functions are used. The bottom of stack may be set anywhere in internal ram by a user program.
The bit addressable internal RAM location 20h is used by the system to hold various software flags. Notice that the individual bits of internal RAM 20h have addresses 0 to 7, 0 being the least significant bit of 20h. The use of each software flag is shown below.

bit	flag name	use
0	dash	set if a dash was detected in the command line
1	equal	set if an equal sign was detected in the command line
2	break	set if a break point is in effect
3	error	set when an error is encountered
4	interrupt	saves the status of EA (EAL) during debug
5	host	set when host mode debugging is selected
6	trace	used internally by the trace routine during debugging
7		reserved for future use

Internal RAM locations 30h to 3Fh are used by the command line processor to save the command line. The parameters extracted from the command line are stored in binary in internal RAM locations 42h to 47h. Internal RAM locations 48h to 4Ch are used by the debug routine. Specifically, location 48h and 49h hold the break point address low and high bytes during debugging. The debug routine returns command to the application program a long jump to the address stored in [49h,49h].
Internal memory use is now summarized.

address	use
20h	software flags

30h..3Fh	command line buffer
42h..47h	buffer for command parameters
48h..4Ch	buffer for break parameters
50h..	stack (RROS does not place more than 16 bytes on stack)

Some important system information is placed at low addresses of ROM. The jump table associated with user accessible system calls is located starting at 100h. ROM locations 400h to 47Fh are reserved for system constants. For example, the ROM version and date is coded as two words (2 bytes each) at locations 400h and 402h respectively. Below is the list of system constants available to the user.

address	use
400h	ROM program version; e.g., 0103h refers to version 1.3
402h	ROM program date; e.g., 1091h refers to October 1991
404h	Contains the end of ROM-based program. Application programs may be placed in EPROM above this address.

If the board is to emulate an (autonomous) embedded controller, rather than branching to the monitor program at reset, control is given to an application program. In this case, the starting address of the application program is placed at locations 480h and 481h, 481h containing the high byte of the start address. If this address is FFFFh, the initialization routine branches to the monitor. If this address is 0000h, the next word at locations 402h and 403h is checked. Similarly, if this word contains an address other than FFFFh or 0000h, a long jump is made to that address. This convention allows an auto-exec routine to be installed by placing its starting address at [401h,400h] or removed by placing 0000 at [401h,400h]. Once zeros are burnt into the EPROM, a new autoexec routine may be installed by placing its starting address at [403h,402h]. This allows for up to 64 such installations, since ROM locations 480h to 4FFh are set aside for autoexec routine addresses.

6 The Interrupt Vector Table

The 8032, 80535, and 80537 have 6, 12, and 14 interrupt sources, respectively. Each interrupt source, when acknowledged, causes a long jump to a fixed location in code memory. The address of this location is referred to as an interrupt vector. The interrupt sources and the corresponding vectors are listed below. The interrupt vectors point to low ROM addresses. The RMB-S redirects these interrupts by placing long jump instructions at the interrupt vector addresses in low ROM.

Source Vector	Redirected to		8052	80535	80537
IE0	0003h	FF00h	x	x	x
TF0	000Bh	FF04h	x	x	x
IE1	0013h	FF08h	x	x	x
TF1	001Bh	FF0Ch	x	x	x
RI(0)+TI(0)	0023h	FF10h	x	x	x
TF2+EXF2	002Bh	FF14h	x	x	x
IADC	0043h	FF18h		x	x
IEX2	004Bh	FF1Ch		x	x
IEX3	0053h	FF20h		x	x
IEX4	005Bh	FF24h		x	x
IEX5	0063h	FF28h		x	x
IEX6	006Bh	FF2Ch		x	x
RI1/TI1	0083h	FF30h			x
CTF	009B	FF34h			x

The system ROM at location 0003h, for example, contains the instruction ljmp FF00h. Similarly, the other interrupt vectors are redirected by long jump instructions.

RROS refers to the memory block FF00h to FF18h as the interrupt vector table. Notice that with the default configuration, the interrupt vector table is in RAM. The initialization routine (init) which is automatically invoked upon reset refreshes the interrupt vector table. This routine is available as a system call. `Init` also places long jump instructions at the interrupt vector table. For example, at location FF00h, corresponding to external interrupt 0 (IE0), `init` places the instruction `ljmp 500h`, where `500h` is the address of the initialization routine invoked at reset. All interrupt vector table entries are similarly initialized with ljmp 500h instructions by the routine init. With the interrupt vector table so initialized, any acknowledged interrupt jumps to start, effectively performing a reset.

Since the interrupt vector table is in RAM, its entries can be modified. Say the interrupt service routine isrIE0 is written and placed in memory. In order to direct IE0 to its service routine, the instruction ljmp isrIE0 is placed in the interrupt vector table, starting at location FF00h. Although the interrupt service routine address may be placed in the vector table by move instructions, it is more convenient to use the system call setintvec. Setintvec is invoked after placing the interrupt service routine address in dptr and the interrupt source, from 0 to 11, in the accumulator.

7 Debug Functions

Debugging a program which has been loaded in RAM may be accomplished by the monitor functions B and K. However, the powerful debugging environment of READ is, in most cases, the preferred way to debug assembly programs. Debugging a program through the use of monitor commands is initiated by selecting a break point in the program. The command B followed by the address of the break point sets the break point. The break point should be placed at the first byte of an instruction. The break point may only be placed at a RAM location. The K command removes or “kills” the break point.

RROS provides minimal debugging utilities through a submenu when an ASCII terminal is being used. Debugging utilities constitute a major portion of RROS. There are two basic modes of debugging: setting break points and tracing, sometimes referred to as single stepping. Each of these modes have advantages and disadvantages. Debugging is geared more toward software development. In terms of hardware debugging, although the debugger offers much help, it cannot track real-time operation issues, such as external hardware interrupts. Such situations call for an in-circuit emulator.

8 How Breakpoints are Handled

A breakpoint is set by replacing three bytes, the byte at the break point and the following two bytes, by a long jump instruction to the break point handler routine in the system ROM. The break point address is stored in internal RAM locations labeled pbuffr and pbuffr+1. The three bytes removed from code are stored in pbuffr+2, pbuffr+3, and pbuffr+4.

Actually, there are two break point handlers, one to be used in conjunction with the IDS, and the other, with an ASCII terminal. Host mode debugging, that is, using the break point handler that works with the IDS, is more powerful than the ASCII terminal mode. Both modes let the user view the internal data RAM. The host mode also allows viewing external memory and modifying internal or external memory. The break point handler in either debugging mode invokes submenus.

The following must be observed when setting a break point.

1. The break point must coincide with the first byte of an instruction in RAM.

2. The program should not make a jump to the byte BP+1 or BP+2 where BP is the break point address.

Point 1 does not pose any restrictions. It is advisable to first obtain a list file from the assembler and then pick appropriate break points.

Although very infrequently, point 2 may require some additional care in placing break points just before labels. Consider the following example.

```
address     instruction       mnemonic
-------------------------------------------
8100        16                      dec  r0
                              begin:
8101        7405                    mov a, #5
.                                   .
.                                   .
.                                   .
8110        80EF                    sjmp begin
```

if a break point is set at 8100, the three bytes at 8100, 8101, and 8102 will be modified to hold a long jump to the break point handler routine, say at address xxxx. That is the byte 16h at 8100 will be modified to 02, and the word 7405 will hold xxxx. Once the break point is placed, when the program execution comes to 8100, the program will branch to the break point handler. However, when the short jump instruction at 8110 is processed, the address xxxx will be fetched and interpreted as an instruction. The recommended way to avoid this is to place nop (no operation) instructions after the dec r0 instruction.

```
address     instruction       mnemonic
-------------------------------------------
8100        16                      dec  r0
8101        00                      nop
8102        00                      nop
                              begin:
8103        7405                    mov a, #5
.                                   .
.                                   .
.                                   .
8112        80EF                    sjmp begin
```

Despite this inconvenience, implementing breakpoints by placing long jump instructions in code has the major advantage that it is not intrusive to the operation of the processor. That is, until the breakpoint is encountered, its presence has no effect on the rest of the program.

9 How Tracing is Handled

Tracing (Single Stepping) is one of the options once a breakpoint is encountered. It is implemented by activating external interrupt 0 (IE0). First the interrupt is chosen to be level activated by clearing TCON.0. Then bit 2 of Port 3 (P3.2), which receives the external signal for external interrupt 0, is cleared. All interrupt priorities are set to the lowest priority by clearing interrupt priority registers IP0 and IP1. By default, IE0 has the highest priority. An interrupt service routine for IE0 is linked by placing its address into the interrupt vector table. Next the break point is removed by restoring the 3 bytes occupied by the call to the break point handler, and the execution resumes from the break point address. Since IE0 is already active, the program jumps to its interrupt service routine after executing one instruction. The interrupt service routine pops the return address, which points to the next instruction, and places a new break point at the next instruction. It

then deactivates IE0, restores the interrupt service routine and returns. Of course, now the instruction upon return is a long jump to the break point handler. Effectively, the processor has executed one instruction and has returned to the break point handler.

The following must be observed when using the trace option.

1. IE0 and P3.2 must not be used by the program.
2. The interrupt service routine inspects the return address and inserts a break point only if the return address is in RAM (8000h-FFFFh). Thus, tracing will not single step through ROM.

Since at each trace instruction the system returns to the break point handler, the user interface is identical to using break points.

10 Debugging with an ASCII Terminal

A break point is set from the monitor prompt (*) using the Bxxxx command and the program run by the Gxxxx command, using an ASCII terminal or the R-Host terminal emulator. When the break point is encountered, the registers followed by a submenu are displayed (sent to the terminal) by the ROM Resident Operating System (RROS). The submenu items are selected by single letter commands. The following submenu items are available.

I displays the contents of the 256 internal RAM locations

R displays the 4 register banks, along with the accumulator, the b register, the stack pointer, the data pointer, and the program counter.

F displays the Special Function Registers. Notice that of the 128 special function registers displayed, not all are used by the processor.

S shows the stack pointer

C removes the break point and continues the program

N removes the break point and sets a new break point at the address which should follow the N command. For example, N8200 sets the new break point at address 8200h.

T branches to the trace utility. Trace places a break point at the next instruction. Thus, by repeatedly issuing the T command, one may single step through the program. At each step, the programmer may examine the state of the processor. Only instructions in RAM can be traced. Thus, T skips over any ROM resident subroutine or user accessible system call which it encounters.

M aborts the program and returns control to the monitor command processor.

Appendix A System Call Source Code

```
; System Calls
;
; Copyright Rigel Corporation, 1990
;
;
; ========================================================
; subroutine ascbin
; this routine takes the ascii character passed to it in the
; acc and converts it to a 4 bit binary number which is returned
; in the acc.
; ========================================================
ascbin:  add   a,  #0d0h        ; if chr < 30 then error
         jnc   notnum
         clr   c                 ; check if chr is 0-9
         add   a,  #0f6h        ; adjust it
         jc    hextry            ; jmp if chr not 0-9
         add   a,  #0ah         ; if it is then adjust it
         ret

hextry:  clr   acc.5             ; convert to upper
         clr   c                 ; check if chr is a-f
         add   a,  #0f9h        ; adjust it
         jnc   notnum            ; if not a-f then error
         clr   c                 ; see if char is 46 or less.
         add   a,  #0fah        ; adjust acc
         jc    notnum            ; if carry then not hex
         anl   a,  #0fh         ; clear unused bits
         ret

notnum:  setb  errorf            ; if not a valid digit
         ret




; ================================================
;  subroutine autoexec start up program
;  scans low rom memory to see if an embedded program
;  is to run at power up.
; ------------------------------------------------
autoexec:
       mov   psw,  #0      ; select register bank 0
       mov   dptr, #ax_tab
ax_0:
       movx  a,   @dptr    ; read low  address
       mov   r0,  a
       inc   dptr
       movx  a,   @dptr    ; read high address
       mov   r1,  a
       anl   a,   r0
       cpl   a
       jz    ax_dn
```



```
        mov   a,   r1
        orl   a,   r0
        jz    ax_1
; ---- start up program specified -------------------
; ---- allow user to abort --------------------------

        push  0            ; lsb to print next
        push  1            ; msb to print next
        lcall crlf
        lcall print
        db   "about to execute program at ",0
        pop   acc      ; recall msb
        lcall prthex
        pop   acc      ; recall lsb
        lcall prthex
        lcall crlf
        lcall print
        db   "hit any key to abort", 0
        clr   ri
        lcall sdelay       ; 2-second grace period
        jb    ri, ax_dn
        lcall sdelay
        jb    ri, ax_dn    ; if key hit the abort...

        lcall crlf         ; ...else start up prog
        pop   acc          ; flush return address ...
        pop   acc          ; ... off from stack
        push  0            ; lsb of start up program
        push  1            ; msb of start up program
        ret                ; effectively jumps to prog

ax_dn: ret
ax_1:  inc   dptr
        ljmp  ax_0
```

```
; ================================================
;  subroutine beep
;  input    : none
;  output   : ^g (bell = 7) sent to serial port
;  destroys : a
; ------------------------------------------------
beep:   push  acc
        mov a, #7h
        lcall sndchr
        pop   acc
        ret
```



```
; ========================================================
; subroutine binasc
; binasc takes the contents of the accumulator and converts it
; into two ascii hex numbers.  the result is returned in the
; accumulator and r2.
; ========================================================
;
binasc:  mov   r2, a             ; save in r2
         anl   a,  #0fh         ; convert least sig digit.
         add   a,  #0f6h        ; adjust it
         jnc   noadj1            ; if a-f then readjust
         add   a,  #07h
noadj1:  add   a,  #3ah         ; make ascii

         xch   a,  r2            ; put result in reg 2
         swap  a                 ; convert most sig digit
         anl   a,  #0fh         ; look at least sig half of acc
         add   a,  #0f6h        ; adjust it
         jnc   noadj2            ; if a-f then re-adjust
         add   a,  #07h
noadj2:  add   a,  #3ah         ; make ascii
         ret
```

```
; ========================================================
; subroutine chkbrk
; this routine checks for the break key. if it is found control
; is passed back to the main monitor loop.
; ========================================================
;
chkbrk:  jnb   ri, nobrk              ; if no chr then return
         mov   a,  sbuf               ; get chr from serial port
         clr   ri                     ; reset rx status bit
break:   cjne  a,  #03h,nobrk        ; if cnt c then
         lcall print                  ; print 'break'
         db 0dh, 0ah," <break> ", 000h
         ljmp  return                 ; return to monitor
nobrk:   ret                          ; else normal return
```



```
; ============================================================
; subroutine crlf
; crlf sends a carriage return line feed out the serial port
; ============================================================
;
crlf:    mov   a,  #0ah         ; print lf
         lcall sndchr
cret:    mov   a,  #0dh         ; print cr
         lcall sndchr
         ret
```

```
; ====================================================
;  subroutine delay - millisecond delay
;   accumulator holds microseconds to delay
;   - 2 microseconds are reserved for the call
;   to this routine.
;  input    : milliseconds to delay in accumulator
;  output   : none
;  destroys : nothing - uses a
; ----------------------------------------------------
; 100h-a6h=5ah=(90)decimal
; 90 * 11 = 990
; plus 10 gives 1 millisecond microsecond
;
;                       microseconds (cycles)
;                      ----------------------
delay:
        dec   a         ;    1

d_olp:
      push acc          ;    2            \
      mov   a, #0a6h    ;    1            |
                        ;                 |
d_ilp:  inc   a         ;    1  \         |
        nop             ;    1  |         |
        nop             ;    1  |         |
        nop             ;    1  |         |
        nop             ;    1  |         |
        nop             ;    1  |- 11     | (acc-1)
        nop             ;    1  | cycles  |- msec
        nop             ;    1  |         |
        nop             ;    1  |         |
        jnz   d_ilp     ;    2  /         |
                        ;                 |
        nop             ;    1            |
        nop             ;    1            |
        nop             ;    1            |
        pop   acc       ;    2            |
                        ;                 |
        djnz  acc,d_olp ;  2            /

; need to wait 998 microseconds more
```



```
        mov   a, #0a6h  ;    1

d_lp2:  inc   a         ;    1  \
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |
        nop             ;    1  |- 11
        nop             ;    1  | cycles
        nop             ;    1  |
        nop             ;    1  |
        jnz   d_lp2     ;    2  /
        nop             ;    1
        nop             ;    1
        nop             ;    1
        nop             ;    1
        nop             ;    1

        ret             ;    2
```

```
; ====================================================
;  subroutine display - display string
;  input    : nibble in accumulator
;  output   : 7-segment pattern in accumulator
;             (acc.0 is segment a, acc.6 is segment g)
;  destroys : a
; ----------------------------------------------------
display:
        inc  a
        movc a, @a+pc
        ret
        db  0c0h  ; 0
        db  0f9h  ; 1
        db  0a4h  ; 2
        db  0b0h  ; 3
        db  99h  ; 4
        db  92h  ; 5
        db  82h  ; 6
        db  0f8h  ; 7
        db  80h  ; 8
        db  90h  ; 9
        db  88h  ; a
        db  83h  ; b
        db  0c6h  ; c
        db  0a1h  ; d
        db  86h  ; e
        db  8eh  ; f
```



```
; ============================================================
; subroutine getbyt
; this routine reads in an 2 digit ascii hex number from the
; serial port. the result is returned in the acc.
; ============================================================
;
getbyt:  lcall getchr            ; get msb ascii chr
         lcall ascbin            ; conv it to binary
         swap  a                 ; move to most sig half of acc
         mov   b,  a             ; save in b
         lcall getchr            ; get lsb ascii chr
         lcall ascbin            ; conv it to binary
         orl   a,  b             ; combine two halves
         ret
```

```
; ============================================================
; subroutine getchr
; this routine reads in a chr from the serial port and saves it
; in the accumulator.
; ============================================================
;
getchr:  jnb  ri, getchr         ; wait till character received
         mov  a,  sbuf           ; get character
         anl  a,  #7fh          ; mask off 8th bit
         clr  ri                 ; clear serial status bit
         ret
```

```
; ============================================================
; subroutine getchrx
; this routine reads in a chr from the serial port and
; saves it in the acc.  it differs from getchr by not
; clearing the most significant bit.
; ============================================================
;
getchrx: jnb  ri, getchrx  ; wait till character received
         mov  a,  sbuf     ; get character
         clr  ri           ; clear serial status bit
         ret
```



```
; ====================================================
;  subroutine inkey
;  input    : peeks at serial port - no wait
;  output   : if chr present
;                then accumulator returns chr
;                else accumulator is 00 (null)
;  destroys : a
; ----------------------------------------------------
inkey:  mov a,  #00h
        jnb ri, doneik
        mov a,  sbuf
        anl a, #7fh
        clr ri
doneik: ret
```

```
; ==========================================================
; subroutine init
; this routine intializes the hardware on the 8051
; ==========================================================
;
init:
           mov psw,  #0h      ; select rb0
           mov dptr, #int0    ; point to interrupt vector table
           mov p2,   #ofsthi  ; offset high byte
           mov r0,   #80h     ; offset into stored rom vector table

transfer:  movx a, @r0        ; transfer...
           movx @dptr, a      ; ...rom interrupt vector...
           inc  r0            ; ...table to...
           inc dptr           ; ...ram (af = 80 + 2f)
           cjne r0, #0afh, transfer

           clr  dashf         ; initialize software flag
           clr  equalf        ; initialize software flag
           clr  breakf        ; initialize software flag
           clr  hostf         ; initialize software flag
; - - - - - - set up serial port - - - - - -
; with a 11.059 Mhz crystal, use timer 1 as the Baud rate generator
; for 9600 Baud

         mov   tmod, #20h     ; set timer 1 for auto reload - mode 2
         mov   tcon, #41h     ; run timer 1 and set edge trig ints
         mov   th1,  #0fdh    ; set timer 1 for 9600 baud with xtal=12mhz
         mov   scon, #50h     ; set serial control reg for 8 bit data
                              ; and mode 1
         ret
```



```
; ================================================
;  subroutine mdelay - millisecond delay
;   delays for 998 microseconds - 2 microseconds are
;   reserved for the call to this routine.
;  input    : none
;  output   : none
;  destroys : nothing - uses a
; ------------------------------------------------
; 100h-a6h=5ah=(90)decimal
; 90 * 11 = 990
; plus 8 gives 998 microseconds
;
;                       microseconds (cycles)
;                      ----------------------
mdelay: push acc       ;    2
        mov   a, #0a6h ;    1

md_olp: inc   a        ;    1  \
        nop            ;    1  |
        nop            ;    1  |
        nop            ;    1  |
        nop            ;    1  |
        nop            ;    1  |- 11  cycles
        nop            ;    1  |
        nop            ;    1  |
        nop            ;    1  |
        jnz   md_olp   ;    2  /

        nop            ;    1
        pop   acc      ;    2
        ret            ;    2
```



```
; ====================================================
;  subroutine percent - converts [0-ff] to [0-99] bcd
;  input    : byte in accumulator
;  output   : [0-99] bcd in accumulator
;
;  destroys : a, r0, dptr, carry flag
; ----------------------------------------------------
percent:
        cjne a, #0ffh, anotff
        mov  a, #99h
        ret

anotff: inc  a
        movc a, @a+pc
        ret
; table of bcd corresponding to the byte
 db 00h, 00h, 01h, 01h, 02h, 02h, 02h, 03h  ;  0 -  7
 db 03h, 04h, 04h, 04h, 05h, 05h, 05h, 06h  ;  8 -  f
 db 06h, 07h, 07h, 07h, 08h, 08h, 09h, 09h  ; 10 - 17
 db 09h, 10h, 10h, 11h, 11h, 11h, 12h, 12h  ; 18 - 1f
 db 12h, 13h, 13h, 14h, 14h, 14h, 15h, 15h  ; 20 - 27
 db 16h, 16h, 16h, 17h, 17h, 18h, 18h, 18h  ; 28 - 2f
 db 19h, 19h, 20h, 20h, 20h, 21h, 21h, 21h  ; 30 - 37
 db 22h, 22h, 23h, 23h, 23h, 24h, 24h, 25h  ; 38 - 3f

 db 25h, 25h, 26h, 26h, 27h, 27h, 27h, 28h  ; 40 - 47
 db 28h, 29h, 29h, 29h, 30h, 30h, 30h, 31h  ; 48 - 4f
 db 31h, 32h, 32h, 32h, 33h, 33h, 34h, 34h  ; 50 - 57
 db 34h, 35h, 35h, 36h, 36h, 36h, 37h, 37h  ; 58 - 5f
 db 37h, 38h, 38h, 39h, 39h, 39h, 40h, 40h  ; 60 - 67
 db 41h, 41h, 41h, 42h, 42h, 43h, 43h, 43h  ; 68 - 6f
 db 44h, 44h, 45h, 45h, 45h, 46h, 46h, 46h  ; 70 - 77
 db 47h, 47h, 48h, 48h, 48h, 49h, 49h, 50h  ; 78 - 7f

 db 50h, 50h, 51h, 51h, 52h, 52h, 52h, 53h  ; 80 - 87
 db 53h, 54h, 54h, 54h, 55h, 55h, 55h, 56h  ; 88 - 8f
 db 56h, 57h, 57h, 57h, 58h, 58h, 59h, 59h  ; 90 - 97
 db 59h, 60h, 60h, 61h, 61h, 61h, 62h, 62h  ; 98 - 9f
 db 62h, 63h, 63h, 64h, 64h, 64h, 65h, 65h  ; a0 - a7
 db 66h, 66h, 66h, 67h, 67h, 68h, 68h, 68h  ; a8 - af
 db 69h, 69h, 70h, 70h, 70h, 71h, 71h, 71h  ; b0 - b7
 db 72h, 72h, 73h, 73h, 73h, 74h, 74h, 75h  ; b8 - bf

 db 75h, 75h, 76h, 76h, 77h, 77h, 77h, 78h  ; c0 - c7
 db 78h, 79h, 79h, 79h, 80h, 80h, 80h, 81h  ; c8 - cf
 db 81h, 82h, 82h, 82h, 83h, 83h, 84h, 84h  ; d0 - d7
 db 84h, 85h, 85h, 86h, 86h, 86h, 87h, 87h  ; d8 - df
 db 87h, 88h, 88h, 89h, 89h, 89h, 90h, 90h  ; e0 - e7
 db 91h, 91h, 91h, 92h, 92h, 93h, 93h, 93h  ; e8 - ef
 db 94h, 94h, 95h, 95h, 95h, 96h, 96h, 96h  ; f0 - f7
 db 97h, 97h, 98h, 98h, 98h, 99h, 99h, 99h  ; f8 - ff
```



```
; ============================================================
; subroutine print
; print takes the string immediately following the call and
; sends it out the serial port.  the string must be terminated
; with a null. this routine will ret to the instruction
; immediately following the string.
; ============================================================
;
print:   pop   dph              ; put return address in dptr
         pop   dpl
         lcall prtstr           ; print string and update dptr
         mov   a,  #1h          ; point to instruction after string
         jmp   @a+dptr          ; return
```

```
; ============================================================
; subroutine prtstr
; this routine takes the string pointed to by the data pointer
; and sends it out the serial port. the string must be
; terminated with a null.
; ============================================================
;
prtstr:  clr  a                     ; set offset = 0
         movc a,  @a+dptr           ; get chr from code memory
         cjne a,  #0h, mchrok       ; if chr = ff then return
         ret
mchrok:  lcall sndchr               ; send character
         inc   dptr                 ; point at next character
         ljmp  prtstr               ; loop till end of string
```

```
; ============================================================
; subroutine prthex
; this routine takes the contents of the acc and prints it out
; as a 2 digit ascii hex number.
; ============================================================
;
prthex:  lcall binasc         ; convert acc to ascii
         lcall sndchr         ; print first ascii hex digit
         mov   a,  r2         ; get second ascii hex digit
         lcall sndchr         ; print it
         ret
```



```
; ========================================================
; subroutine prsphx
; this routine first prints a space then takes the contents of
; the acc and prints it out as a 2 digit ascii hex number.
; ========================================================
;
prsphx:  mov   r2, a          ; save acc in r2
         mov   a,  #20h       ; print space
         lcall sndchr
         mov   a,  r2         ; recall acc
         lcall prthex         ; print it
         ret




; ====================================================
;  subroutine sdelay - second delay
;   delays for 999998 microseconds - 2 microseconds
;   are reserved for the call to this routine.
;  input    : none
;  output   : none
;  destroys : nothing - uses a
; ----------------------------------------------------
; 100h-91h=6fh=(111)decimal
; 9008 * 111 = 999888
; plus 102 from second loop
; plus 8 gives 999998 microseconds
;
;                       microseconds (cycles)
;                      ----------------------
sdelay: push acc              ;   2
        mov  a, #91h          ;   1

sd_olp: inc   a      ; \
        lcall mdelay ; |
        lcall mdelay ; |
        lcall mdelay ; |
        lcall mdelay ; |
        lcall mdelay ; |
        lcall mdelay ; |
        lcall mdelay ; |- loop takes 9008 microseconds
        lcall mdelay ; |
        lcall mdelay ; |
        nop          ; |
        nop          ; |
        nop          ; |
        nop          ; |
        nop          ; |
        jnz   sd_olp ; /

        mov   a, #33h         ;   1
sd_ilp: djnz  acc, sd_ilp     ; -loop takes 2*33h=66h=(102)dec

        pop   acc             ;   2
        ret                   ;   2
```



```
; ============================================================
;  subroutine setintvec - set interrupt vector
;  input    : a contains interrupt source [0..11]
;             dptr contains isr address
;  output   : none
;  destroys : a, dptr
; ------------------------------------------------------------
setintvec:
        push dpl            ; push isr address
        push dph
        anl  a, #0fh        ; just to be sure
        rl   a              ; multiply by 4
        rl   a
        mov  dph, #0ffh     ; ram vector table
        mov  dpl, a         ; dptr points to vector table
        mov  a, #2          ; ljmp instruction
        movx @dptr, a
        inc  dptr
        pop  acc            ; pop isr address high byte
        movx @dptr, a
        inc  dptr
        pop  acc            ; pop isr address low byte
        movx @dptr, a       ; new int vector placed
        ret
```

```
; ============================================================
; subroutine sndchr
; this routine takes the chr in the acc and sends it out the
; serial port.
; ============================================================
;
sndchr:  clr  scon.1             ; clear the tx  buffer full flag.
         mov  sbuf,a             ; put chr in sbuf
txloop:  jnb  scon.1, txloop     ; wait till chr is sent
         ret
; ================================================
```



```
; ====================================================
; The following are not part of the system ROM
; To use, include them in your source code
; ====================================================

;  subroutine kbdclrw -
;              clear keyboard and wait for keypressed
;  input    : none
;  output   : none
;  destroys : nothing - uses a
; ----------------------------------------------------
kbdclrw:
        push  acc
kcw_0:
        jnb   ri, kcw_1
        clr   ri
        sjmp  kcw_0
kcw_1:
        lcall getchr
        pop   acc
        ret




; ====================================================
;  subroutine kbdwait - waits for keypressed
;  input    : none
;  output   : none
;  destroys : nothing - uses a
; ----------------------------------------------------
kbdwait:
        push  acc
        lcall getchr
        pop   acc
        ret
```



```
; ============================================================
; subroutine getword
; this subrouinte reads in a string in ascii from the serial
; port.  the line must be terminated with a <cr>.  the line is
; stored in the line buffer. the maximum line length is 16
; bytes including the <cr>.  the word is returned in the first
; parameter buffer at location pbuffr.
; ============================================================
;
getword:
       push  psw
       mov   psw, #0
       push  3
       push  4
       push  5
       push  6

       clr   errorf
       clr   ri               ; flush serial port
       mov   r0,  #lbuffr     ; init line buffer ram to 0's
gwinit:
       mov   @r0, #0ffh
       inc   r0
       cjne  r0,  #lbuffr+11h, gwinit

       mov   r4,  #0h         ; init line buffer count (pointer)
gw0:   mov   a,   r4          ; if line length > 15 then error
       jnb   acc.4, gwok0
       ljmp  gwerr_ret

gwok0: lcall getchr           ; read in chr
       mov   r1, a            ; save in r1
       cjne  r1, #7fh, gwnodel      ; if del then del
       sjmp  gwdel
gwnodel:
       cjne  r1, #08h, gwnorub      ; if back space then del
gwdel: mov   a,  r4           ; do not back up over prompt
       jz    gw0
       mov   a,  #08h         ; backspace
       lcall sndchr
       mov   a,  #20h         ; send a space
       lcall sndchr
       mov   a,  #08h         ; backspace
       lcall sndchr
       dec   r4               ; dec line buffer pointer.
       sjmp  gw0
gwnorub:
       mov   a, r1            ; recall char
       clr   c
       add   a, #0c0h
       jnc   gw_num           ; if alphabetic then
       mov   a, r1            ; then make upper case
       clr   acc.5
       mov   r1, a
gw_num:
       mov   a,  r4           ; save chr in line buffer.
       add   a,  #lbuffr      ; compute address
```



```
        mov   r0, a
        mov   a,  r1            ; get chr
        mov   @r0,a
        lcall sndchr            ; echo character

        inc   r4                ; point to next location in buffer
        cjne  r1, #0dh, gw0     ; if not cr then return
        mov   a,  #0ah          ; if chr=cr then line feed and do normal ret
        lcall sndchr
        mov   a,  r4            ; if cr only then error
        cjne  a,  #1h, gwok
        ljmp  gwerr_ret
gwok:
     mov   r0, #lbuffr-1        ; point before first chr in line buffer

; - - - - - -  read in parameter  - - - - - - - - - - -
gw1:
        inc   r0                ; point to next location in line buffer
        mov   a,  @r0           ; get chr from line buffer
        cjne  a,  #0dh, gw2     ; if cr then end
        ljmp  gwerr_ret         ; return on error

gw2:    cjne  a,  #20h, gw3     ; if chr=space then loop
        sjmp  gw1
gw3:
        mov   r1, #pbuffr       ; parameter 0
        lcall getparx           ; get parameter from line buffer
        jnb   errorf, gwret

gwerr_ret:
        setb  errorf
        lcall subbadpar

gwret: pop   6
        pop   5
        pop   4
        pop   3
        pop   psw
        ret
```

