H8/300 tools

Embedded Software Development: H8/300 tools

Application Note

Balu Donthi

HITACHI®

AE-0508
August 1994

INTRODUCTION

An embedded computer system is a "hidden computer" within a "system". An embedded system often consists of a micro-computer and peripheral hardware which can be used to perform certain specific tasks. The "system" may be a home appliance, automotive, telephone, network controller, toy, elevator, etc. The applications are innumerable and endless. It is not a computer system in the traditional context, i.e., where a software program can be developed on the computer and can be used to control itself. Therefore, the software program to control an embedded system has to be developed on a "traditional" computer system.

This application note provides a tutorial on the usage of Hitachi software development tools for the H8/300 microcontroller family and explains the customization that has to be made to different software routines to control the embedded computer.

The Hitachi software development tools for the H8/300 microcontroller family consists of a C compiler, Assembler, Linker, Librarian & Simulator Debugger. The application program shown in this application note has been customized to execute on the H8/300 series simulator debugger.

Following software tools have been used for the tutorial in this application note.

CH38	C Cross Compiler	v2.0B
ASM38	Macro Assembler	v3.2E
LNK	Linker	v5.1
LBR	Librarian	v1.2B
SD38	Simulator Debugger	v2.4

Software Development Issues: Embedded Systems Vs Native Systems

Following customization has to be made when developing software for embedded systems. In a traditional computer system (native development environment) these operations are usually performed by the operating system:

- The reset vector for the micro-controller should be setup so that it points to the entry point in the program.
- The interrupt vectors for the micro-controller have to be setup.
- The stack and heap spaces required for the application program should be allocated and the stack pointer should be initialized to the stack area.
- The variables altered by the program should be located in the RAM and the constant variables should be located in the ROM.
- If the application program uses the run-time libraries then the low level routines have to be customized for the I/O operations.

In a native development environment the software developer does not have to worry about any of the above operations since the operating system in the computer takes care of performing these operations. Figure.1 gives a pictorial representation of the program flow in an embedded system.

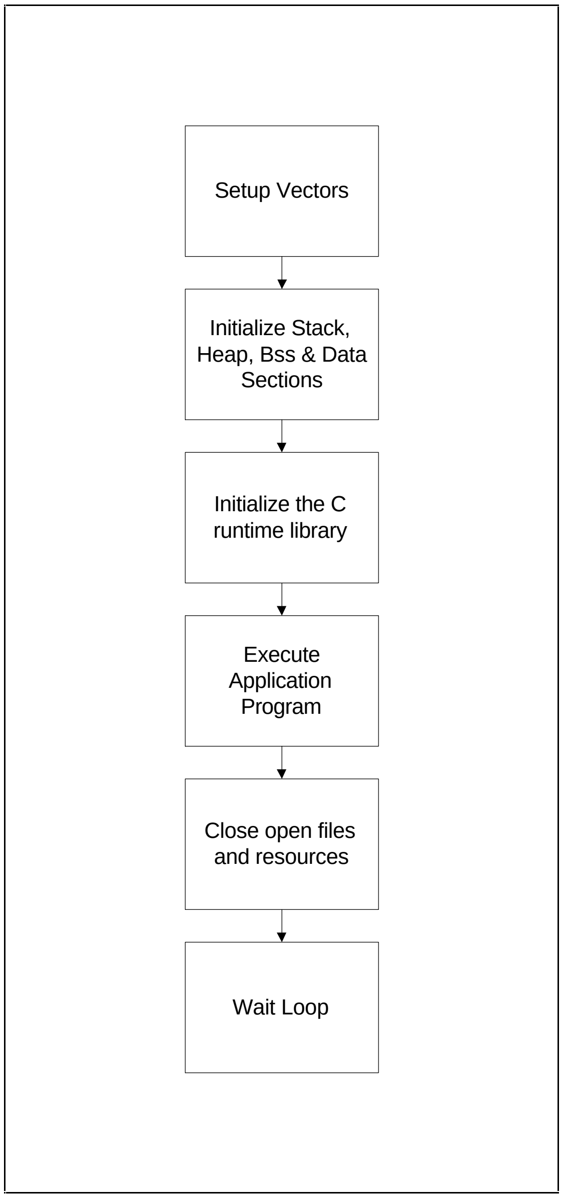

Setup Vectors
Initialize Stack, Heap, Bss & Data Sections
Initialize the C runtime library
Execute Application Program
Close open files and resources
Wait Loop

Figure.1 Program Flow in an Embedded System

Sections generated by the compiler

The compiler normally generates the following sections when a C program is compiled, they are:

P - Program or Code section

C- Constants section

D- Initialized Data section (located in ROM)

B- Uninitialized Data (or BSS) section

The other sections that are needed for the application program have to be created in the startup routine. In this tutorial, various sections needed for the application i.e. vect, stack, heap and the "ram" data section (Section-R) are created in the startup and other initialization routines. The startup routine copies the variable data located in ROM (Section D generated by the compiler) to the RAM (Section R) at the application startup time.

Initializing Vectors

In an embedded system the microcontroller vectors should be initialized first. The H8/300 micro-controller has 48 bytes of memory reserved for vectors space, i.e., for reset vectors, illegal instructions, IRQs, Interrupts and other on chip peripherals. The vectors are located at address H'0000. In the example initialization code, only the reset vectors are initialized but there is space allocated for the user to setup vectors that are used by the application. Refer to the file "vectbl.src" for the vector table initialization.

Sample vector initialization:

```
.SECTION VECT,DATA,LOCATE=H'0000
.IMPORT __ENTRY
.DATA.W  ENTRY   ;Power On Reset PC
.DATA.W (STARTOF STACK) + (SIZEOF STACK)
; Power On Reset SP
.DATA.W __ENTRY ; Manual Reset PC
.DATA.W (STARTOF STACK) + (SIZEOF STACK)
; Manual Reset SP
```

Initialization of Stack

The stack area is used each time a function is called and is deallocated when the function returns. The stack pointer should be initialized before calling a subroutine or any high level C function. The compiler does not create a STACK section so the user has to define a section, allocate stack space and initialize the Stack pointer. The reset vector points to the location __ENTRY and this label is defined in the file "start.src". In the sample application 2Kb of stack is allocated. This number can be increased or decreased based on the application.

Example assembly code for section declaration and call to the initialization routines are shown below:

```
.SECTION STACK, STACK, ALIGN=2
.RES.B H'800    ; 2K stack
.SECTION P, CODE, ALIGN=2
__ENTRY:
MOV.W   #(STARTOF STACK) + (SIZEOF
STACK), R15 ; Initialize stack (SP)
MOV.W   #INIT, R2
JMP     @R2
NOP
.END
```

Also, in the file start.src there is a constants section which has the start address and the size of Data section (D) in ROM, Data section (R) in RAM and the BSS section which is the uninitialized data section. At this point the thread of execution is transferred to the routine _INIT.

Initialization of Data Sections

The function _INIT is the high level function which calls all the initialization functions, application routine and close routine. Once the application has been executed the _CLOSEALL() routine closes all the files, resources and than waits in a loop for a hardware reset.

Source for the function _INIT:

```
void _INIT(void)
{
        _INITSCT();
        _INITLIB();
        main();
        _CLOSEALL();
        for(;;);
}
```

The uninitialized data section (B) has to be initialized to 0 before program execution according to the C language specifications. This operation is performed through the startup routine. The initialized data section (D) contains data with initial values. After linking the application program these initial values are located in the ROM. These initial values are modified by the application program therefore they have to be located in the RAM before program execution. These initial values are modified by the application program therefore they have to be located in the RAM before program execution. The function _INITSCT() accesses the starting addresses and size of the uninitialized data initializes it to 0. The function _INITSCT() also copies the initial values from the data section "D" to data section "R".

Initializing the data for the Run Time Library

The standard C function library is included with the compiler, if the application program uses any of these functions than some of the data used by the runtime libraries has to be initialized. If the application program does not use any of the standard C library functions than this code (_INITLIB()) can be eliminated to make the application program smaller in size. The function _INITLIB() initializes the error checking variable "errno" to 0. This variable can be checked for successful execution of library functions. If the call to a library function has been successful than errno has 0 and if it did not complete successfully than this value is set to 1.

The _INITLIB() function calls _INIT_IOLIB() and _INIT_OTHERLIB functions. As the name suggests the _INIT_IOLIB function initializes _iob data structure which is used by functions like PRINTF, SCANF, FOPEN, FCLOSE, etc. If the application program does not use any of the standard library functions for the input/output operations than these initializations may not be performed. The _iob structure is defined in the stdio.h header file and the function _INIT_IOLIB() is defined in the init.c file.

Sample initialization of _iob structure:

```
/* Clears buffer */
fp -> _bufptr = NULL;
/* Clears buffer counter */
fp -> _bufcnt = 0;
/* Clears buffer length */
fp -> _buflen = 0;
```

```
/* Clears base pointer */
fp -> _bufbase = NULL;
/* Clears I/O flags */
fp -> _ioflag1 = 0;
fp -> _ioflag2 = 0;
fp -> _iofd = 0;
```

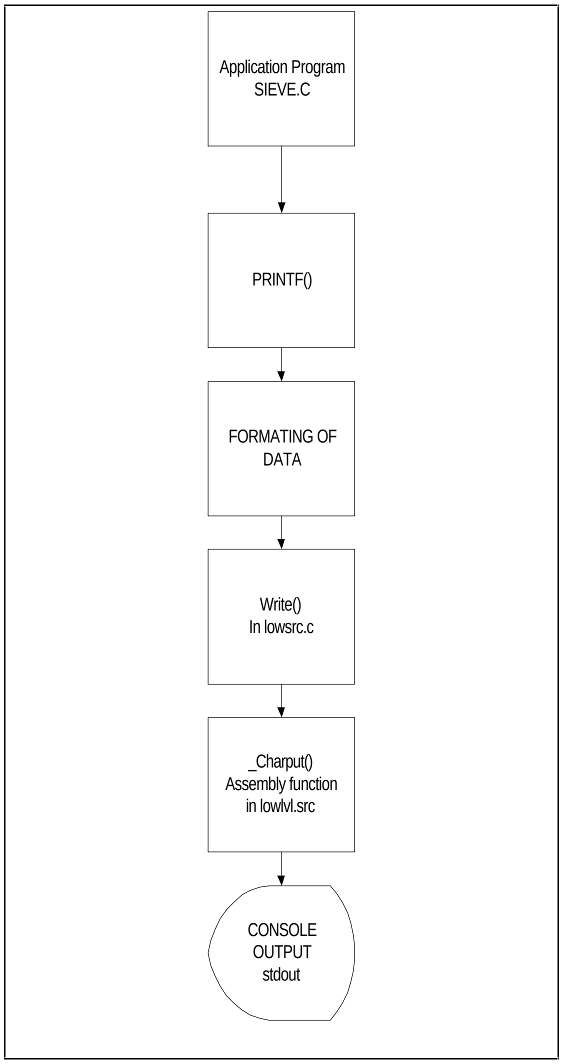
The function _INIT_IOLIB() also initializes the standard input (stdin) for "No data buffering", and "disabled file access". The standard output (stdout) and standard error (stderr) are also opened and initialized for "no data buffering".

The _INIT_OTHERLIB() function sets the initial value of random number generator function (rand()) to 0 and also sets the pointer (_s1ptr) used in the strtok() (converts strings to tokens) to 0. This initialization need not be performed if the rand() and strtok() functions are not used by the application.

Application Program "sieve.c"

The application program used for our tutorial is sieve.c. The sieve.c program is Eratosthenes Sieve prime number calculation program. It has been scaled down with MAX_PRIME set to 17 instead of 8091. This program uses the printf runtime library function to display these prime numbers on the console (stdout). The printf routine also uses various resources like heap, other library routines for formatting the data and finally calls the "write" routine. The write routine is not supplied with the runtime library and has to be written by the user. In this tutorial a sample write() function is provided in the lowsrc.c file. The write routine also calls the _charput() function which outputs a character to the console (stdout). The _charput routine has to be customized to suit the hardware requirements. However, for this tutorial the _charput routine has been modified to work with the SDSH simulator debugger.

The lowsrc.c has the low level I/O functions which can be customized by the user. These low level functions are used by the C runtime library functions. Also, present in the lowsrc.c file is the read() function which is called by Input functions in the runtime library like scanf(). The read function calls the _charget() routine, this routine is also customized to work with the SD38 simulator debugger. Both the routines _charput and _charget are written in assembly language and can be found in the lowlvl.src file. The code for initializing the heap section can be found in the lowsrc.c file.



Program Flow for printing to the CONSOLE

Closing the Files & Resources

After the execution of the application is over the _CLOSE() function is called to close all the opened resources and files. This function is located in the lowsrc.c file, it accesses the _iob structure to locate all the opened files and resources than closes them. Once all the files are closed the program goes into a wait loop waiting for a hardware reset.

Creating a CPU information file

A CPU information file has to be created for the SD38 debugger before loading the application program. This file can be created by using the CIA utility. Using this utility you can setup the CPU type, address bus size, data bus size, memory wait states and memory map.

The utility can be invoked by typing

```
cia sieve.cpu<cr>
/* Select H8/300 CPU by typing */
? 3<cr>
/* Select 16 bit Address Bus */
BIT SIZE 16 ? :16
/* Enter a comment */
COMMENT?:CPU  information  file  for
   Sieve program.
/* Setup  a RAM area for simulation */
*** MAP MENU ***
0:ROM 1:EXTERNAL 2:RAM 3:I/O 4:EEPROM
   .:END
?2
/* Setup RAM from 0 thru H'ffff */
*RAM  AREA  START  ADDRESS?  0000  END
   ADDRESS?  ffff
/* Set the RAM wait state to 0 */
STATE COUNT ?  1
/* Set the Data Bus Size to 8 */
DATA BUS SIZE?  8
/* Exit from the CIA utility */
*  RAM AREA START ADDRESS? . <cr>
```

Building the Application Program

A batch file sieve.bat has been provided with this tutorial which compiles, assembles and links the startup files with the application program. **Before executing the batch file the installation location should be determined and the environment variables needed for the software tools should be set.**

The environment variable **CH38** must point to the directory where all the header files supplied with the compiler are located. The **CH38TMP** environment variable should point to a disk with enough space (depends on the application program). The **HLNK_LIBRARY1** environment variable is used by the linker and it should point to one of the run-time libraries supplied with the compiler. **HLNK_LIBRARY2** and **HLNK_LIBRARY3** environment variables are available for use with the user libraries. All the directories containing the compiler, assembler and simulator debugger executables should be specified in the DOS path. The default installation location for the H8/300 tools is c:\hitachi, if the tools are installed in this directory than the following environment variables should be set.

```
set CH38=c:\hitachi\ch38\include
set CH38TMP=c:\
set HLNK_LIBRARY1=c:\hitachi\ch38\lib
\ch38reg.lib
set PATH=c:\hitachi\ch38\bin;
         c:\hitachi\asm38;
         c:\hitachi\sd38;
```

Batch file sieve.bat for building the application:

```
sieve.bat
ch38 /debug sieve.c
shc /debug init.c
asm38 start.src /debug /cpu=300
asm38 lowlvl.src /debug /cpu=300
lnk -sub=sieve.cmd
```

The invocation of the compiler is "ch38" and the "/debug" option is necessary to include all the symbol information in the output object. The invocation of the assembler is "asm38" and the "/debug" option is required for including the symbol information in the object file. The default output by the compiler is a re locatable object & the optimizations are ON by default. To get a summary of all the available compiler options, invoke the compiler without any options i.e. "ch38". The output of the assembler is also a re locatable object so all the objects are linked using the linker "lnk".

A linker command file is provided with the tutorial, it has all the commands needed to perform the link operations.

sieve.cmd

/* This command is necessary for symbol information in output */

```
debug
```

/* Specifies linker output to be a absolute file i.e. SYSROF object */

```
form a
```

/* Loads the sieve.obj into linker */

```
input sieve.obj
```

/* Loads the start.obj into linker */

```
input start.obj
```

/* Loads the init.obj into linker */

```
input init.obj
```

/* Loads the lowlvl.obj into linker */

```
input lowlvl.obj
```

/* Creates a new section for Initialized data */

```
rom (D,R)
```

/* Locates const section at 0x1000 & stack section at 0x9000*/

```
start P,D,C(1000),B,R,STACK(9000)
```

/* Specifies the starting point of application to be at the label __ENTRY */

```
entry __ENTRY
```

/* Specifies the output of linker to be sieve.abs */

```
output sieve
```

/* Specifies the linker map file to be sieve.map */

```
print sieve
```

/* Starts the linking operation and exits out of the linker after the link is over */

```
EXIT
```

Executing the Application Program

All the library functions have been customized so that they can be executed with the H8/300 Simulator Debugger SD38. Please go through the following steps for getting familiar with the various simulator debugger commands and running the application program.

1) Invoke the debugger by typing

```
sd38 /cpu=sieve.cpu sieve.abs
```

2) Set the trap for Simulated I/O

```
trap_address h'FE
```

3) Load the application program

```
load sieve.abs
```

4) Display the application map.

```
map
```

5) Display the C source code.

```
da 1000
```

6) Single Step through the C code.

```
s
```

7) Display the micro-controller registers.

```
register
```

8) Look at the symbol information

```
symbol
```

9) Execute the application program

```
go
```

At this point the application program starts executing and the prime numbers are printed on the console. The program goes into a loop after execution. To break the program out of the loop, please type ^t.

For complete information on the usage of compiler, assembler and simulator debugger tools please refer to the respective user manuals.

Appendix   Source Listings for the Startup files and Application Program

Listing 1. **start.src** (begin)

```
;/*************************************************************************/
;/* File:                 start.src                                      */
;/* Description: Sets the stack pointer and calls _INIT function       */
;/************************************************************************/
; The following section is needed for initializing the vars section &
; clearing the non-initialized section

         .SECTION D,DATA,ALIGN=2
         .SECTION R,DATA,ALIGN=2
         .SECTION B,DATA,ALIGN=2
         .SECTION C,DATA,ALIGN=2

__D_ROM .DATA.W (STARTOF D)             ; Start address of section D
__D_BGN .DATA.W (STARTOF R)             ; Start address of section R
__D_END .DATA.W (STARTOF R)+ (SIZEOF D) ; End address of section R
__B_BGN .DATA.W (STARTOF B)             ; Start address of section B
__B_END .DATA.W (STARTOF B) + (SIZEOF B) ; End address of section B

         .EXPORT __D_ROM
         .EXPORT __D_BGN
         .EXPORT __D_END
         .EXPORT __B_BGN
         .EXPORT __B_END
         .EXPORT __ENTRY
         .IMPORT __INIT

         .SECTION STACK, STACK, ALIGN=4
         .RES.B H'800    ; 2K stack

         .SECTION P, CODE, ALIGN=2

__ENTRY:
         MOV.W  #(STARTOF STACK) + (SIZEOF STACK), R7 ;Initialize stack (SP)
         MOV.W   #__INIT, R2
         JMP     @R2
         NOP
         .END
```

Listing 1.**start.src** (end)

Listing 2. **vectbl.src** (begin)

```
;/*********************************************************************/
;/* File: VECTBL.SRC                                                  */
;/*  Initailizes the vector table                                     */
;/*                                                                   */
;/*********************************************************************/
      .SECTION STACK,STACK
      .SECTION VECT,DATA,LOCATE=H'0000
      .IMPORT __ENTRY
reset   .data.w  __ENTRY      ;Power On Reset PC, vect 0
resrv1  .data.w  0      ;vect 1
resrv2  .data.w  0      ;vect 2
pervec  .res.w H'37     ; reserved for other peripheral vectors

      .end
```

Listing 2. **vectbl.src** (end)

Listing 3. **INIT.C** (begin)

```
/**********************************************************************/
/* File:                  INIT.C                                     */
/* Description:  Main application function, performs initializations,   */
/*               calls application and closes files and waits for reset */
/**********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#define _NFILE 20

extern char *_s1ptr;
extern void srand(unsigned int);
void _CLOSEALL(void);
extern void main(void);
void _INITSCT(void);
void _INITLIB(void);
void _INIT_IOLIB(void);
void _INIT_OTHERLIB(void);
/* Declares FILE-type data in the C language */
```

```
extern int *D_ROM, *B_BGN, *B_END, *D_BGN, *D_END;
void _INIT(void)
{
        _INITSCT();
        _INITLIB();
        main();
        _CLOSEALL();
        for(;;);
}


void _INITLIB(void)
{
        errno=0;

        _INIT_IOLIB();
        _INIT_OTHERLIB();

}

void _INIT_IOLIB(void)
{
        FILE *fp;
                        /* Initialize FILE-type data */
        for ( fp =_iob; fp<_iob+_NFILE; fp++)
        {

                fp -> _bufptr = NULL;
                fp -> _bufcnt = 0;
                fp -> _buflen = 0;
                fp -> _bufbase = NULL;
                fp -> _ioflag1 = 0;
                fp -> _ioflag2 = 0;
                fp -> _iofd = 0;
        }

        /* Opens standard I/O file */

        if (freopen( "stdin" , "r",  stdin) == NULL) /* Opens standard input
file */

                stdin->_ioflag1=0xff;              /* Disables file access */
                stdin->_ioflag1 |= _IOUNBUF ;       /* No data buffering */


        if (freopen( "stdout", "w", stdout) == NULL) /*Opens standard output
file */

                stdout -> _ioflag1=0xff;
                stdout -> _ioflag1 |= _IOUNBUF ;
        if (freopen( "stderr", "w", stderr) == NULL) /* opens standard error
file */

                stderr -> _ioflag1 = 0xff;
                stderr -> _ioflag1 |= _IOUNBUF;


}

  void _INITSCT(void)
  {
        int *p, *q;
        /* Non-initialized area is initialized to zeros */

        for (p =_B_BGN; p <= _B_END; p++)
        {
        *p=0;
        }
        /* Initialized data is copied from ROM to RAM   */
        for (p =_D_BGN, q = _D_ROM; p <= _D_END; p++, q++)
        {
        *p = *q;
        }
}


void _INIT_OTHERLIB(void)
{
        srand(1);   /* Sets initial value when rand function is used */
        _s1ptr=NULL; /*Initializes the pointer used in the strtok function*/
}

void _CLOSEALL(void)
{
        int i;

        for (i=0; i < _NFILE; i++)

                if(_iob[i]._ioflag1 & ( _IOREAD | _IOWRITE |_IORW))

                fclose( &_iob[i]);
}

/*************************************************************************/
/*                              lowsrc.c:                               */
/*----------------------------------------------------------------------*/
/*              H8/300-series simulator debugger interface routine
*/
/*      - Only standard I/O files (stdin, stdout, stderr) are supported  */
/*************************************************************************/
#include <string.h>
/* file number */
```

```
#define STDIN 0                 /* Standard input (console)            */
#define STDOUT 1                /* Standard output (console)           */
#define STDERR 2                /* Standard error output (console)     */

#define FLMIN 0                 /* Minium file number                  */
#define FLMAX 3                 /* Maximum number of files             */

/* file flag */

#define O_RDONLY 0x0001         /* Read only                            */
#define O_WRONLY 0x0002         /* Write only                           */
#define O_RDWR   0x0004         /* Both read and write                  */

/* special character code */

#define CR 0x0d                 /* Carriage return                      */
#define LF 0x0a                 /* Line feed                            */

/* size of area managed by sbrk */

#define HEAPSIZE 1024

/*************************************************************************/
/* Declaration of reference function                                     */
/* Reference of assembly program in which the simulator debugger input of */
/* ouput characters to the console                                       */
/*************************************************************************/

extern void __charput(char);      /* One character input
*/
extern char __charget(void);     /* One character output
*/

/*************************************************************************/
/* Definition of static variable:                                        */
/* Definition of static variables used in low-level interface routines   */
/*************************************************************************/

char flmod[FLMAX];              /* Open file mode specification area     */

static union {
               long dummy;       /* Dummy for 4-byte boundary            */
               char heap[HEAPSIZE];/*Declaration of the area managed by
sbrk*/

              }heap_area ;

static char *brk=(char *)&heap_area;/*End address of area assigned by sbrk*/

/*************************************************************************/
/*      open:file open                                                   */
/*              Return value: File number (Pass)                         */
/*                            -1          (Failure)                      */
/*************************************************************************/
int open(char *name,            /* File name                            */
        int mode)               /* File mode                            */
{
        /* Check mode depending on file name and return file numbers    */
        if(strcmp(name,"stdin")==0) {   /* Standard input file          */
                if((mode&O_RDONLY) == 0)
                        return -1;
                flmod[STDIN]=mode;
                return STDIN;
        }

        else if(strcmp(name,"stdout")==0) { /* Standard output file     */
                if((mode&O_WRONLY)==0)
                        return -1;
                flmod[STDOUT]=mode;
                return STDOUT;
        }

        else if(strcmp(name,"stderr")==0) { /* Standard  error file      */
                if((mode&O_WRONLY)==0)
                        return -1;
                flmod[STDERR]=mode;
                return STDERR;
        }

        else
                return -1;
}

/*************************************************************************/
/*      close: File close                                                */
/*             Return value: 0 (Pass)                                    */
/*                          -1 (Failure)                                 */
/*************************************************************************/

int close(int fileno)                   /* File number                   */
{
        if(fileno<FLMIN || FLMAX<fileno) /* File number range check      */
                return -1;

        flmod[fileno] = 0;
        return 0;                       /* File mode reset               */

}

/*************************************************************************/
/* read:Data read                                                        */
/*      Return value:Number of read characters  (Pass)                   */
/*                      -1                      (Failure)                */
/*************************************************************************/
```

```
int read(int fileno,            /* File number                          */
        char *buf,              /* Destination buffer address
*/
        unsigned int count)     /* Number of read characters
*/
{
        unsigned int i;

/* Check mode according to file name and stores each character in buffer  */

        if(flmod[fileno] & O_RDONLY || flmod[fileno] & O_RDWR) {
                for(i=count; i>0; i--) {
                        *buf=charget();
                        if(*buf==CR)  /*Line feed character replacement */
                                *buf=LF;
                                buf++;
                        }
                        return count;
                }
                else
                        return -1;
}

/**********************************************************************/
/* write: Data Write                                                  */
/*      Return value:Number of write characters (Pass)                */
/*             -1                           (Failure)                 */
/**********************************************************************/
int write(int fileno,                          /* File number          */
          char *buf,                    /* destination buffer address */
          unsigned int count)           /* Number of write characters */
{
        unsigned int i;
        char c;
/* Check mode according to file name and output each character     */

        if(flmod[fileno] &O_WRONLY || flmod[fileno] &O_RDWR) {
                for(i=count; i>0; i--) {
                        c=*buf++;
                        charput (c);
                        }
                        return count;
        }
        else
                return -1;
}

/**********************************************************************/
/*      lseek : Definition of file read/write position                 */
/*           Return value:offset from the top of file read/write position */
/*                              -1              (Failure)              */
/*      (lseek is not supported in the console input/output)          */
```

```
/***************************************************************************/
long lseek(int fileno,                         /* File number
*/
             long offset,                        /* Read/Write position
*/
             int base)                           /* Origin of offset
*/
{
         return -1;
}

/***************************************************************************/
/*      sbrk:Data write                                                    */
/*              Return value: Start addresss of the assigneed area (Pass) */
/*                               -1                              (Failure)*/
/***************************************************************************/
char *sbrk(unsigned long size)          /* Assigned area size
*/

{
     char *p;

     if(brk+size>heap_area.heap+HEAPSIZE) /* Empty area size
*/
                  return (char *) -1;

     p=brk ;
     brk += size;
     return p;

}
```

Listing 3. **INIT.C** (end)

Listing 4. **SIEVE.C** (begin)

```
/*********************************************************************/
/* sieve.c -- Eratosthenes Sieve prime number calculation              */
/* scaled down with MAX_PRIME set to 17 instead of 8091                */
/*************************************************************************/
#include <stdio.h>
#define MAX_ITER        1
#define MAX_PRIME       17

char    flags[MAX_PRIME];

main ()
{
        register int i,k;
        int     prime,count,iter;

        for (iter = 1;iter<=MAX_ITER;iter++)
                {
                count = 0;
                for(i = 0; i<MAX_PRIME; i++)
                        flags[i] = 1;
                for(i = 0; i<MAX_PRIME; i++)
                        if(flags[i])
                                {
                                prime = i + i + 3;
                                k = i + prime;
                                while (k < MAX_PRIME)
                                        {
                                        flags[k] = 0;
                                        k += prime;
                                        }
                                count++;
                                printf(" prime %d = %d\n", count, prime);
                                }
                }
        printf("\n%d primes\n",count);
}
```

Listing 4. **SIEVE.C** (end)

Listing 5. **LOWLVL.SRC** (begin)

```
;--------------------------------------------------------------------------
;
; lowlvl.src
;
; H8/300-series simulator debugger interface routines to input or output a
; single character.
;
;--------------------------------------------------------------------------
;--------------------------------------------------------------------------
;
; lowlvl.src
;
; 300 series simulator debugger interface routines to input or output a
; single character.
;
;--------------------------------------------------------------------------
         .CPU 300
         .EXPORT _charput
         .EXPORT _charget
         .EXPORT __INIT_LOWLEVEL

SIM_IO: .EQU    H'00FE          ; Trap address

         .SECTION        P, CODE, ALIGN=2

;
; this routine may differ for different environments and here is used
; as a dummy
;
__INIT_LOWLEVEL:
         RTS
         NOP

;-------------------------------------------------------------------------
;
; _charput: single character output
;           C interface: charput(char)
;
;-------------------------------------------------------------------------

_charput:
         MOV.W   #A_DATA,R4     ; Address of Data
         MOV.B   R0L,@R4        ; char parameter is passed in R0L
         MOV.W   #A_PARM,R1     ; Pointer to Address of Data
         MOV.W   R4,@R1         ; Initializing Pointer to Address of Data
```

```
          MOV.W   @putc,R0        ; H'0102 is moved into R0
          MOV.W   #SIM_IO,R2      ; H'FE is moved into R2
          JSR     @R2             ;Delayed branching, outputs a char to
                                   ;console
          NOP
          RTS

_charget:                        ; Gets 1 character from console
          MOV.W   #A_PARM,R1      ; Pointer to Address of Data
          MOV.W   #A_DATA,R0      ; Address of Data
          MOV.W   R0,@R1          ; Initializing Pointer to Address of Data
          MOV.W   @getc,R0        ; H'0101 system call addr into R0
          MOV.W   #SIM_IO,R2      ; H'FE is moved into R2
          JSR     @R2             ; Delayed branching, gets a char from
                                 ;console
          NOP                     ;
          MOV.W   #A_PARM,R1      ;
          MOV.W   @R1,R0          ;
          MOV.B   @R0,R0L         ;
          RTS                     ;
          NOP                     ;

                   .ALIGN  4
A_DATA:         .DATA.W DATA
A_PARM:         .DATA.W PARM
A_FNO:          .DATA.W FILENO
F_putc:         .DATA.W H'0108
F_getc:         .DATA.W H'0107
putc:           .DATA.W H'0102  ; outputs 1 Character to simulated output
getc:           .DATA.W H'0101  ; gets 1 character from console, sim input
;--------------------------------------------------------------------------
;
; I/O buffer destination
;
;--------------------------------------------------------------------------

          .SECTION        B,DATA,ALIGN=2

PARM:   .RES.W  1
FILENO: .RES.B  1
DATA:   .RES.B  1

          .END
```

Listing 5. **LOWLVL.SRC** (end)

Listing 6. **SIEVE.BAT** (begin)

```
REM /*******************************************************************/
REM /* File:                       SIEVE.BAT                          */
REM /* Description: Batch file to build the SIEVE Application         */
REM /*               to execute on the H8/300 simulator Debugger
        */
REM /*******************************************************************/

ch38 /debug sieve.c
ch38 /debug init.c
asm38 start.src /debug /cpu=300
asm38 lowlvl.src /debug /cpu=300
lnk /sub=sieve.cmd
```

Listing 6. **SIEVE.BAT** (end)

Listing 7. **SIEVE.CMD** (begin)

```
;/*********************************************************************/
;/* File:                 SIEVE.CMD                                 */
;/* Description: Links all the relocatable objects for sieve application */
;/*********************************************************************/
debug
form a
input sieve.obj
input start.obj
input init.obj
input lowlvl.obj
rom (D,R)
start P,D,C(1000),B,R,STACK(9000)
entry __ENTRY
output sieve
print sieve
EXIT
```

Listing 7. **SIEVE.CMD** (end)