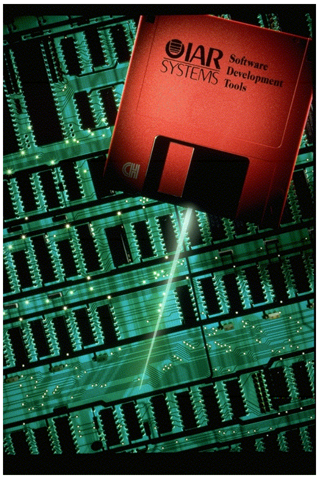
ICCH83

INTEGRATED C COMPILER

HITACHI H8/300 DEVELOPMENT TOOLS



INTEGRATED ENVIRONMENT

The ICCH83 toolset is delivered as a complete toolset with C compiler, assembler, linker, librarian and run-time libraries. To help reduce development times and make the tools easier to use the delivery also includes a menu driven user interface with mouse control. This user interface also includes an error-sensitive editor and make utilities. Use the IAR Integrated Environment - and get to market faster.

The IAR ICCH83 development kit offers the choice of C to H8/300 and H8/300H applications, from single-chip to banked design.

ICCH83 implements the full ANSI C language, and provides extended keywords specific to the H8/300 architecture. With its built-in chip-specific optimizer, the IAR H8/300 compiler generates very efficient and reliable PROMable code.

Combined with fully comprehensive documentation, the IAR ICCH83 gets you started on your H8/300 project in no time, making the learning process fast and easy. In addition to a solid technology, our professional technical support is yet another reason engineers adopt IAR C.

COMPILER

Full ANSI C compatibility
The IAR H83 C Compiler is fully compatible with the ANSI C standard. All data types required by ANSI are supported without any exceptions (see figure 1). *float* and *double* are represented in the IEEE 32- or 64-bit precision. Bitfields are based on *char*, *short* or *long* datatypes making port manipulation very efficient.

Full ANSI C compatibility means that the IAR C Compilers follow not only the ANSI syntax but also the less well known requirements that ANSI puts on run-time behavior such as integral promotions and precision in floating point calculations to name two specific and important areas.

DATA TYPE	SIZE (bytes)	VALUE RANGE
bit	1 bit	0 or 1
sfr	1	0 to 255
sfrp	2	0 to 65535
signed char	1	-128 to +127
unsigned char	1	0 to 255
short & int	2	-32768 to +32767
unsigned short & int	2	0 to 65535
signed long	4	-2^{31} to $2^{31}-1$
unsigned long	4	0 to $2^{32}-1$
float IEEE 32-bit	4	±1.18E-38 to ±3.39E+38, 7 digits
double IEEE 64-bit	8	±2.23E-308 to ±1.79E+308, 16 digits
pointer	1,2,4	object address

Figure 1 *Data representation supported by the IAR H83 C Compiler.*

H83 Specific extensions

To ideally suit development for embedded systems, standard C needs additional functionality. IAR Systems has defined a set of extensions to ANSI C, specific to the H8/300 architecture (see Figure 2). All of these extended keywords can be invoked by using the *#pragma* directive, which maintains compatibility with ANSI and code portability.

In Addition there is also a set of intrinsic functions that are specially designed for H8/300 (see Figure 2). These functions maps to assembler instructions that can be directly invoked in C code as a function call. The intrinsic functions shown in the table are only some of the available functions.

Efficient floating point

The compiler comes with full floating point support. It follows the IEEE 32-bit representation using an IAR Systems proprietary register based algorithm, which makes floating point manipulation extremely fast.

TYPE	KEYWORD	DESCRIPTION
Function	interrupt	Creates an interrupt function that is called through an interrupt vector. The function preserves the register contents and the processor status.
	monitor	Turns off the interrupts while executing a monitor function.
	non_banked	Declares a non banked function.
	tiny_func	Called indirectly via an exception vector.
	near_func	Access range from 0H to FFFFH.
	far_func	Unrestricted access to 16MB range.
	banked_func	Used in banked switching mode.
	C_task	Inhibits register saving (used in real-time kernel applications).
	ANSI_main	Forec main() to save registers.
Variable	no_init	Puts a variable in the no_init segment. Does not get intialized at start-up.
	tiny	Data object stored in the tiny segment. Access using 8-bit addressing.
	near	Data object stored in the near segment. Access using 16-bit addressing.
	far	Data object stored in the far segment. Access using 32-bit addressing. Object size <64KB.
	huge	Data object stored in the huge segment. No restrictions on size.
Segment	codeseg	Renames the CODE segment.
	constseg	Creates a new CONST segment.
	dataseg	Creates a new DATA segment.
Intrinsic	sleep	Executes the SLEEP instruction.
	no_operation	Executes the NOP instruction.
	read_e_port	Reads a byte from an address using MOVFPE.
	write_e_port	Writes a byte to an address using MOVTPE.
	disable_max_time	Sets maximum interrupt disable time.
	do_byte_eepmov	Copy a sequence of bytes to an EPROM using EEPMOV.B.
	do_word_eepmov	Copy a sequence of bytes to an EPROM using EEPMOV.W.
	func_stack_mask	Gets a pointer to correct function return address.
	set_interrupt_mask	Sets the interrupt priority level.
	read_ccr	Reads the CCR register.
	write_ccr	Writes to the CCR register.
	and_ccr	ANDs to the CCR register.
	or_ccr	ORs to the CCR register.
	xor_ccr	Exclusive-ORs to the CCR register.

Figure 2 *IAR Systems embedded C extensions.*

Processor mode		Extra small	Tiny	Mini	Small	Large	Banked
Function calls	64KB Mode	tiny_func	tiny_func	banked_func	near_func	near_func	banked_func
Function calls	1 MB & 16 MB Mode	far_func	far_func	banked_func	far_func	far_func	banked_func
Data pointers	64KB Mode	near	near	near	near	near	near
Data pointers	1 MB & 16 MB Mode	far	far	far	huge	huge	huge
Stack size	64KB Mode	256 bytes	64 KB	256 bytes	256 bytes	64 KB	64 KB
Stack size	1 MB & 16 MB Mode	256 bytes	64 KB	64 KB	64 KB	16 MB	16 MB
Intrinsic calls		Could be selected as tiny_func or far_func under any mode or via a compiler switch.					

Figure 3. *Memory models.*

Memory models for any hardware design

Every design has its own memory requirements. The ICCH83 compiler has two sets of six different memory models allowing a best fit selection(see Figure 3).

ASSEMBLER

Macro-Assembler for time-critical routines

The IAR C Compiler kit comes with a relocatable structured assembler. This provides the option of coding time-critical sections of the application in assembly without losing the advantages of the C language. The preprocessor of the C compiler is incorporated in the assembler, thus allowing use of the full ANSI C macro language, with conditional assembly, macro definitions, if statements, etc. C include files can also be used in an assembly program. All modules written in assembly can easily be accessed from C and vice versa, making the interface between C and assembly a straightforward process.

Powerful Set of Assembler Directives

The assembler provides an extensive set of directives to allow total control of code and data segmentation. Directives also allow creation of multiple modules within a file, macro definitions and variable declarations.

LINKER

The IAR XLINK Linker supports complete linking, relocation and format generation to produce H8/300 PROMable code (see Figure 4).

The XLINK generates over 30 different output formats and is compatible with most popular emulators and EPROM burners.

The XLINK is extremely versatile in allocating any code or data to a start address, and checking for overflow. Detailed cross reference and map listing with segments, symbol information, variable locations, and function addresses are easily generated.

Examples of linker commands	Description
-Z seg_def	Allocates a list of segments at a specific address.
-F format_name	Selects one of more than 30 different absolute output formats.
-x -l file_name	Generate a map file containing the absolute addresses of modules, segments, entry points, global/static variable, and functions.
-D symbol=value	Define a global symbol and equates it to a certain value.

Figure 4. *Example of different linker commands*

LIBRARIAN

The XLIB Librarian creates and maintains libraries and library modules. Listings of modules, entry points, and symbolic information contained in every library are easily generated.

XLIB can also change the attributes in a file or library to be either conditionally or unconditionally loaded, i.e. loaded only if referred to or loaded without being referred to.

ANSI C LIBRARIES

The IAR C Compiler kit comes with all libraries required by *ANSI free standing implementation of C*. Additionally, ICCH83 comes with low-level routines required for embedded systems development (see Figure 5).

C LIBRARY FUNCTIONS
DIAGNOSTICS<assert.h> assert
CHARACTER HANDLING<ctype.h> isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit tolower, toupper
VARIABLE ARGUMENTS<stdarg.h> va_arg, va_end, va_list, va_start
NON LOCAL JUMPS<setjmp.h> longjmp, setjmp
INPUT/OUTPUT<stdio.h> getchar, gets, printf, putchar, puts, scanf, sscanf, sprintf
GENERAL UTILITIES<stdlib.h> abort, abs, atof, atol, atoi, bsearch, calloc, div, exit, free, labs, ldiv, malloc, rand, realloc, srand, strtod, strtol, strtoul, qsort
STRING HANDLING<string.h> memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcoll, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok, strxfrm
MATHEMATICS<math.h> acos, asin, atan, atan2, ceil, cos, cosh, exp, exp10, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh
LOW-LEVEL ROUTINES<iccbutl.h> _formatted_write, _formatted_read

Figure 5. *Library functions. IAR C Compiler comes with all libraries required by ANSI.*

UTILITIES & EXTRAS

User interface, editor and Make utility installation is easy and straight forward due to the installation program which will check for other IAR installations. ICCH83 comes with a mouse-controlled menu-driven user interface that includes an error-sensitive ASCII editor. An easy-to-use Make utility is also integrated in the interface environment.

SUPPORT & UPDATES

IAR H83 toolkit comes with the following benefits:

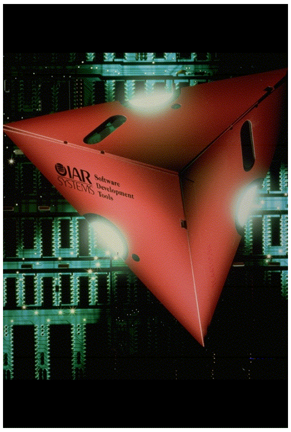
- Updates released within 90 days after purchase free of charge.
- On-line free technical support.

HOSTS

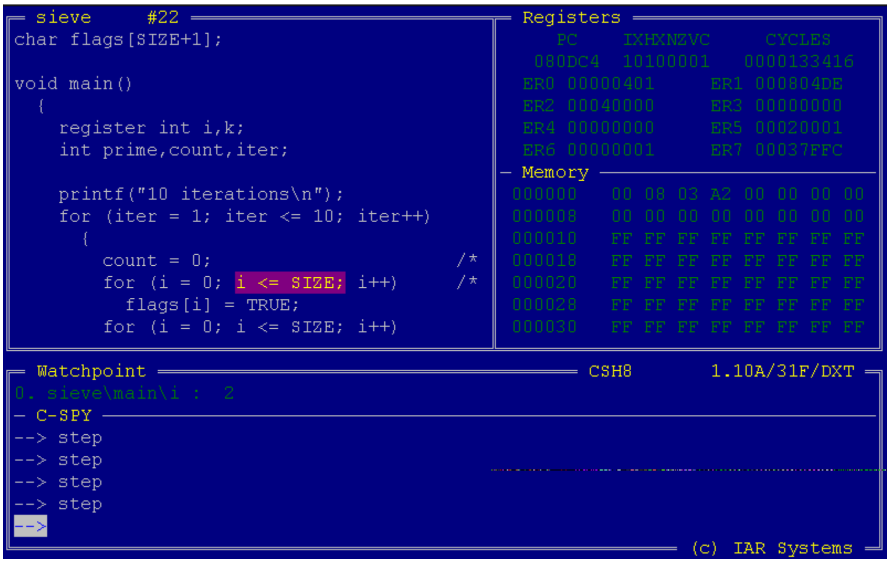
- IBM PC and compatibles. Minimum 386, DOS 4.x, and 4 MB of RAM.
- Windows 3.1x, 95 and NT 3.51 or later in a DOS window.
- SUN 4 (SPARC): SUN-OS, Solaris.
- HP 9000/700: HP-UX.

C-SPY H83 SIMULATOR/DEBUGGER

FOR HITACHI H8/300



The IAR H83 C-SPY is a high level language simulator/debugger. C-SPY combines the detailed control of execution needed for embedded development debugging with the flexibility and power of the C language.



```
 sieve      #22                                 Registers
char flags[SIZE+1];                               PC      IXHXNZVC      CYCLES
                                                080DC4    10100001    0000133416
void main()                                     ER0  00000401      ER1  000804DE
  {                                             ER2  00040000      ER3  00000000
    register int i,k;                           ER4  00000000      ER5  00020001
    int prime,count,iter;                       ER6  00000001      ER7  00037FFC
                                               Memory
    printf("10 iterations\n");                  000000    00 08 03 A2 00 00 00 00
    for (iter = 1; iter <= 10; iter++)          000008    00 00 00 00 00 00 00 00
      {                                         000010    FF FF FF FF FF FF FF FF
        count = 0;                        /*    000018    FF FF FF FF FF FF FF FF
        for (i = 0; i <= SIZE; i++)       /*    000020    FF FF FF FF FF FF FF FF
          flags[i] = TRUE;                      000028    FF FF FF FF FF FF FF FF
        for (i = 0; i <= SIZE; i++)             000030    FF FF FF FF FF FF FF FF

 Watchpoint                                      CSH8        1.10A/31F/DXT
0. sieve\main\i :  2
 C-SPY
--> step
--> step
--> step
--> step
-->
                                                          (c)  IAR Systems
```

USER INTERFACE

Short learning curve

C-SPY is a window-oriented simulator/debugger which provides a friendly and easy-to-navigate debugging environment.

No set-up problems

C-SPY does not need to be set-up to offer powerful debug features. All functionality is present from start-up. The C-SPY screen could be reduced to only two windows (Source and Command) for simplicity or be divided into the following user-selectable windows:

C/ASM source code. Displays source code on C or assembly levels, and highlights the line being executed. Allows placement of breakpoints directly on the C or ASM source line.

Registers. Displays register contents and the cycle count.

Memory. Simulates memory space of the cpu. Displays the content of a user selectable address range, ROM, RAM or stack.

Watchpoint. Displays the content of variables and expressions. Globals, locals, structures, arrays, and pointers are all supported.

Terminal I/O. A unique C-SPY feature where the screen becomes the output and the keyboard becomes the input. A very useful feature for debugging embedded applications when logical flow is of interest or the target is not yet ready.

A Powerful Command Set

A powerful yet easy to use command set; includes all that is needed for embedded debugging environments. Frequently used commands are invoked via function keys.

Built-in Assembler & Disassembler

In addition to modifying variables and symbol values, C-SPY H83 also provides the flexibility of modifying the code during a debugging session. This feature is often needed while debugging embedded applications.

CONTACT INFORMATION

USA
IAR Systems Inc.
One Maritime Plaza
San Francisco,
CA 94111
Tel: +1 415-765-5500
Fax: +1 415-765-5503
Email: info@iar.com

SWEDEN
IAR Systems AB
P.O. Box 23051
S-750 23 Uppsala
Tel: +46 18 16 78 00
Fax: +46 18 16 78 38
Email: info@iar.se

GERMANY
IAR Systems GmbH
Brucknerstrasse 27
D-81677 Munich
Tel: +49 89 470 6022
Fax: +49 89 470 9565
Email: info@iar.de

UK
IAR Systems Ltd.
9 Spice Court,
Ivory Square
London SW11 3UE
Tel: +44 171 924 3334
Fax: +44 171 924 5341
Email: info@iarsys.co.uk

Home Page: http://www.iar.se