hitex
DEVELOPMENT TOOLS

Revision F

Company X
SOFTWARE QUALITY MANAGEMENT
‘C’ COMPILER REPORT
KEIL C166 v3.12j

APPROVAL	NAME	SIGNATURE	DATE
Author		______________________	__________
Software Engineering		______________________	__________
Project Manager		______________________	__________
Validation Manager		______________________	__________
Engineering Director		______________________	__________

(Please destroy all earlier versions)

Contents

1 INTRODUCTION 5
1.1 Overview 5
1.2 Disclaimer 5

2 ITEMS ADDRESSED BY THIS REPORT 6
2.1 Quality Control & Validation 6
2.1.1 Quality Systems 6
2.1.2 Year 2000 Issues 6
2.1.3 Compiler Testing Prior To Release 6
2.1.4 Empirical Measures Of Compiler Integrity 6
2.1.5 Compiler Problem Status 7
2.2 Standard Functions 7
2.2.1 Standard ISO-C library functions not provided 7
2.2.2 Standard ISO-C library functions that differ in prototype from what would be expected 8
2.2.3 Standard ISO-C functions behaving in a non-standard manner 11
2.3 Implementation-Specific Issues 11
2.3.2 The layout of storage for parameters. 13
2.3.3 How a diagnostic is identified. 17
2.3.4 What constitutes an interactive device. 19
2.3.5 The number of bits in a character in the execution character set. 20
2.3.6 The result of casting a pointer to an integer or vice versa. 21
2.3.7 What constitutes an access to an object that has volatile-qualified type. 21
2.3.8 The maximum number of declarators that may modify an arithmetic, structure or union type. 21
2.3.9 Whether the value of a single-character character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Whether such a character constant may have a negative value. 22
2.3.10 The mapping of source file character sequences 22
2.3.11 The null pointer constant to which the macro NULL expands. 22
2.3.12 CTYPE.H Functions 22
2.3.13 The values returned by the mathematics functions on domain errors. 24
2.3.14 Compiler Actions With Unusually Terminated Source Files 28
2.3.15 The outcome when the result of an integer arithmetic function (*abs*, *div*, *labs*, or *ldiv*) cannot be represented 30
2.3.16 The outcome when an *lvalue* with an incomplete type is used in a context that requires the value of the designated object. 31
2.3.17 Mismatches in type between *lvalue* and object 31
2.3.18 Incomplete Data Declarations 33
2.3.19 Volatile/non-volatile mismatches 34
2.3.20 Incomplete Types And Tentative Declarations 35
2.3.21 Shift Left And Shift Right Operations 35
2.3.22 Integer Division Behaviour 36
2.3.23 Floating Point Truncation 36
2.3.24 Float To Integer Conversions 37
2.3.25 Type promotion, `char` To `int` 40
2.3.26 Type Promotion From int To `long` 40
2.4 Compiler System Controls Which Impact Software Integrity And Maintainability 41
2.4.1 Overview 41
2.4.2 Compiler Controls 42
2.4.3 Linker Controls 44
2.4.4 CPU Configuration Controls In START167.A66 44
2.4.5 Characteristics Of HLARGE Model Programs 44
2.4.6 The Special Sections "?C_CLRMEMSEC" And "?C_INITSEC". 48
2.5 Floating Point Library Test 49
2.5.1 Preliminary Work 49
2.5.2 Preliminary Results 49
2.5.3 Discussion Of Results 50
2.6 Limits Of The Keil C166 Compiler System 51
2.6.1 Compiler Implementation Limits 51
2.6.2 Siemens/Keil OMF66 Object Module Format Limits 51

3 COMPOSITION OF THE COMPILER TOOLKIT 52

4 APPENDIX A 54
4.1.1 Moving The USERSTACK On-Chip 54

5 APPENDIX B 58
5.1 Pointer Casting And Conversions 58

5.1.1 Casting From Basic Types To Pointers 58

6 APPENDIX C 60
6.1 Pointers In The C166 Compiler 60
6.2 The Most Common Pointers In C166 60
6.2.1 `far` Pointers 60
6.2.2 `huge` Pointers 60
6.2.3 `xhuge` Pointers 60
6.2.4 `sdata` Pointers 60
6.3 Summary Of Pointer Declarations 60
6.4 Special Note On #pragma MOD167 For C167 CPUs 61

7 APPENDIX D 62
7.1 BSI Definition Of Year 2000 Compliance – Keil C166 Tools 62

8 APPENDIX E 66
8.1 Results Log Of PARANOIA Test For KEIL C166 v3.12j 66
8.2 Results Log Of PARANOIA Test For Microsoft C v11.00 70

9 APPENDIX F 75
9.1 Plum-Hall test summary files. 75
9.1.1 Invocation Batch File 75
9.1.2 Summary File 1 76
9.1.3 Summary File 2 79
9.1.4 Explanation Of Error Reports From The Plum Hall Test 81

10 APPENDIX G 82
10.1 Known Problems In C166 v3.12j 82

Revision History

New Revision	Description Of Changes
A	Preliminary version
B	Preliminary version with register and userstack usage described
C	Typographic corrections made and section on library functions added.
D	Retesting with v3.12j. Compiler & OMF66 limits added. PARANOIA test repeated with single precision
E	Addition of v3.12j problem list.
F	Addition of Plum Hall error descriptions
G	Change to divide by zero integer division results to indeterminate

1 Introduction

1.1 Overview

The XXX product range has been designed using devices from the Siemens C16x family of microcontrollers. Upon the advice of Hitex (UK) Ltd., the Keil C166 ‘C’ compiler toolkit was selected to support software development.

This report has been prepared to address the issues raised in the Company X ‘C’ Software Coding Requirements Revision X.X. Its purpose is to:

- Provide evidence of the adequacy of the compiler toolkit for this purpose.
- Provide details of how the compiler implements those features that are either not specified by the ISO/ANSI ‘C’ standard, or are explicitly identified as specific to the implementation of the compiler. This will enable to software development and validation personnel to be aware of items that may not be portable to other platforms.

Where reference is made to the ISO/ANSI ‘C’ Standard, this shall be taken to mean “ISO/IEC 9899:1990 Programming Languages – C” 1990 with Technical Corrigendum 1995 as published by ISO.

1.2 Disclaimer

Whilst every effort has been made to ensure the accuracy and completeness of this report, neither Hitex (UK) Ltd. or the author can be held liable for any consequences resulting from errors, omissions or interpretation of information presented therein.

2 Items Addressed By This Report

2.1 Quality Control & Validation

Evidence shall be provided that the C compiler has been developed to the requirements of a quality management system that meets the requirements of ISO 9001:1994 and that its performance is satisfactory, having passed tests executed using a recognised validation suite (e.g. Plum-Hall).

Details of known problems with the version of the compiler used shall be provided, along with details of how such information can be kept up to date as new releases become available.

2.1.1 Quality Systems

Keil do not have an accredited ISO9000 quality management system, as is common in the embedded compiler field. Hitex is certified to ISO EN 9001 08/94, certificate number report Q1 01 97 02 27639 001 and Keil Elektronik GmbH as a major supplier since 1989, has been fully assessed as part of this process.

2.1.2 Year 2000 Issues

The tested 3.12j compiler is Year 2000 compliant, as defined by BSI PD2000 (appendix D)

2.1.3 Compiler Testing Prior To Release

The Keil C166 compiler is tested using the Plum-Hall C validation suite prior to release. The Keil Plum-Hall licence number is CVS245. A listing of the results of the test are given in appendix F.

Note: "Plum-Hall" is a registered trademark of Plum-Hall Inc., USA.

2.1.4 Empirical Measures Of Compiler Integrity

The Keil C166 compiler is in use at approximately 300 UK locations. Based on UK usage, on average one minor bug is reported and confirmed once every 12 weeks across approximately 300 active users of C166 v3.05a to 3.12j. The average time to fix a bug is three weeks from confirmation of the problem.

Keil has been represented in the UK by Hitex since 1989 and is a tier-one supplier.

2.1.5 Compiler Problem Status

By special arrangement with Keil, a list of known problems for the tested release can be made available. Such information for C166 v3.12j can be found in appendix G.

2.2 Standard Functions

2.2.1 Standard ISO-C library functions not provided

The Keil C166 compiler contains a subset of the ANSI library function set that is appropriate for use in a deeply embedded C167 environment. It omits the following specific and classes of library functions:

- Stream IO functions, fflush, fputc, fprintf,
- All file-orientated functions fopen(), fclose(),
- All signal functions
- All error handling functions
- All time and date functions
- All environment communication functions
- Type-generic functions

List Of ISO-C Library Functions Not Implemented In Keil C166

`abort`	`fgets`	`ldexp`	`strerror`
`asctime`	`fmod`	`ldiv`	`strftime`
`assert`	`fopne`	`localeconv`	`strstr`
`atexit`	`fprintf`	`localtime`	`strtod`
`bsearch`	`fputc`	`mblen`	`strok`
`clearerr`	`fputs`	`mbstowcs`	`strol`
`clock`	`fread`	`mktime`	`stroul`
`ctime`	`freopen`	`perror`	`strxfrm`
`difftime`	`frexp`	`putc`	`system`
`div`	`fscanf`	`qsort`	`time`
`exit`	`fseek`	`raise`	`tmpfile`
`fclose`	`fsetpos`	`remove`	`tmpnam`
`feof`	`ftell`	`rename`	`ungetc`
`ferror`	`fwrite`	`reweind`	`vfprintf`
`fflush`	`getc`	`setlocale`	`wcstombs`
`fgetc`	`getenv`	`setvbuf`	`wctomb`
`fgetpos`	`gmtime`	`strcoll`	

2.2.2 Standard ISO-C library functions that differ in prototype from what would be expected

2.2.2.1 Type Of `size_t`

`size_t` is the unsigned integer type result of the sizeof operator. It is implemented as a typedef in `stddef.h`.

Example

```
typedef unsigned int size_t;
```

2.2.2.2 Keil C166 String Handling Function Prototypes

The string handling functions whose prototypes are listed in string.h differ slightly from the ISO standard in that the source pointer does not have the `const` attribute. `const` objects in the C166 compiler are defined as being located in a read-only area such as ROM. The limitation of source strings only being located in ROM would be impracticable in a real C167 system. Where a parameter or return value is of type size_t, C166 gives the type as `unsigned int`.

```
extern int  strpos   (char *s, char c);
extern char *strcat (char *s1, char *s2);
extern char *strncat (char *s1, char *s2, unsigned int n);
extern int strcmp (char *s1, char *s2);
extern int strncmp (char *s1, char *s2, unsigned int n);
extern char *strcpy (char *s1, char *s2);
extern char *strncpy (char *s1, char *s2, unsigned int n);
extern size_t strlen (char *);
extern char *strchr  (char *s, char c);
extern char *strrchr (char *s, char c);
extern int  strrpos  (char *s, char c);
extern int  strspn   (char *s, char *set);
extern int  strcspn  (char *s, char *set);
extern char *strpbrk (char *s, char *set);
extern char *strrpbrk(char *s, char *set);
```

Note: The passing of source and destination pointers the above library functions with an attribute other than `near` will result in a WARNING which must not be ignored.

2.2.2.3 Additional Keil C166 String Functions

C166 provides additional versions of the ISO-C string functions that are intended for those cases where the source and destination pointers are of type `far`. These are:

```
extern char far *fstrcat  (char far *s1, char far *s2);
extern char far *fstrncat (char far *s1, char far *s2, unsigned int n);
extern int fstrcmp  (char far *s1, char far *s2);
extern int fstrncmp (char far *s1, char far *s2, unsigned int n);
extern char *fstrcpy  (char far *s1, char far *s2);
extern char *fstrncpy (char far *s1, char far *s2, unsigned int n);
extern size_t fstrlen (char far *);
extern char far *fstrchr  (const char far *s, char c);
extern char far *fstrrchr (const char far *s, char c);
extern int  fstrpos  (const char far *s, char c);
extern int  fstrrpos (const char far *s, char c);
extern int  fstrspn    (char far *s, char far *set);
extern int  fstrcspn   (char far *s, char far *set);
extern char *fstrpbrk  (char far *s, char far *set);
extern char *fstrrpbrk (char far *s, char far *set);
```

Note: The 'f' prefix does not imply that these functions are file-orientated. Such functions are not available in C166.

2.2.2.4 ISO-C String Handling Function Prototypes

```
char *strcpy(char * s1, const char * s2);
char *strncpy(char * s1, const char * s2, size_t n);
char *strcat(char * s1, const char * s2);
char *strncat(char * s1, const char * s2, size_t n);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
size_t strspn(const char *s1, const char *s2);
size_t strcspn(const char *s1, const char *s2);
char *strpbrk(const char *s1, const char *s2);
```

2.2.2.5 Unimplemented String Functions In C166

```
int strcoll(const char *s1, const char *s2);
size_t strxfrm(char * s1, const char * s2, size_t n);
char *strstr(const char *s1, const char *s2);
char *strtok(char * s1, const char * s2);
char *strerror(int errnum);
```

2.2.2.6 Keil C166 Memory Copying Functions

C166 memory copying functions differ from ISO-C versions in that source pointer does not have the `const` attribute. Where a parameter or return value is of type size_t, C166 gives the type as `unsigned int`.

```
extern int   memcmp  (void *s1, void *s2, unsigned int n);
extern void *memcpy  (void *s1, void *s2, unsigned int n);
extern void *memchr  (void *s,  char val, unsigned int n);
extern void *memccpy (void *s1, void *s2, char val, unsigned int n);
extern void *memmove (void *s1, void *s2, unsigned int n);
extern void *memset  (void *s,  char val, unsigned int n);
```

Additional versions of these functions are provided that cater for those situations where memory regions are to be handled that are larger than the default location qualifier for the current memory model permits.   These functions are listed below:

```
extern      int  fmemcmp  (void far *s1, void far *s2, unsigned int n);
extern void far *fmemcpy  (void far *s1, void far *s2, unsigned int n);
extern void far *fmemchr  (void far *s,  char val, unsigned int n);
extern void far *fmemccpy (void far *s1, void far *s2, char val, unsigned int n);
extern void far *fmemmove (void far *s1, void far *s2, unsigned int n);
extern void far *fmemset  (void far *s,  char val, unsigned int n);
extern      int    xmemcmp  (void xhuge *s1, void xhuge *s2, unsigned long n);
extern void xhuge *xmemcpy  (void xhuge *s1, void xhuge *s2, unsigned long n);
extern void xhuge *xmemchr  (void xhuge *s,  char val, unsigned long n);
extern void xhuge *xmemccpy (void xhuge *s1, void xhuge *s2, char val, unsigned long n);
extern void xhuge *xmemmove (void xhuge *s1, void xhuge *s2, unsigned long n);
extern void xhuge *xmemset  (void xhuge *s,  char val, unsigned long n);
```

2.2.2.7 ISO-C Memory Copying Functions

```
void *memcpy(void * s1, const void * s2 size_t n);
void *memmove(void *s1, const void *s2, size_t n);
char *strcpy(char * s1, const char * s2);
char *strncpy(char * s1, const char * s2, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
void *memchr(const void *s, int c, size_t n);
void *memset(void *s, int c, size_t n);
```

2.2.2.8 Input/Output Functions

Example Of `gets()`

```
char *gets(char *s);
```

The ISO-C `gets` function reads characters from the input stream pointed to by `stdin`, into the array pointed to by s, until END-OF-FILE is encountered or a NEWLINE character is read. Any NEWLINE character is discarded, and a null character is written immediately after the last character read into the array

Example Of The Keil C166 `gets()`

```
char *gets (char *, unsigned int n);
```

The Keil C166 `gets` function calls the `getchar` function to read a line of characters into a string *s. The line consists of all characters up to and including the first NEWLINE character ('\n'). The NEWLINE character is replaced by a null character ('\0') in the string. The parameter n specifies the maximum number of characters that may be read. If n characters are read before ENWLINE is encountered, the `gets` function terminates the string with the null character ('\0') and returns.

2.2.3 Standard ISO-C functions behaving in a non-standard manner.

2.2.3.1 Memory Allocation Functions

Memory management functions assume that the non-ISO C `init_mempool()` function has been previously called with parameter consisting of the address of a RAM area which can be used for the heap plus its overall size. `malloc()`, `calloc(), realloc()` can then access blocks in this area.

Note**:** These functions should not be used in a safety-critical system

2.2.3.2 Formatted And Other IO Functions

`printf()` directs its formatted output to `putchar()` (contained in PUTCHAR.C) which by default addresses C167 serial port 0.

`scanf()` obtains it input characters from `_getkey()` (contained in GETHAR.C) which be default addresses C167 serial port 0.

`gets()` is covered in section 2.2.2.8

`puts(const char *s)` writes a string followed by a NEWLINE character ('\n') to the output stream using the putchar function.

`putchar(char c)` transmits the input character c to the C167 serial port zero. It is common for this function to be modified to allow characters to be directed to other output devices such as an LCD panel or alternative serial port. The source code for the `putchar` incorporated in the Keil runtime library set can be found in "\C166\LIB\PUTCHAR.C".

`getchar()` reads a single character from the input stream using the `_getkey()` function. The character read is passed back to the `putchar()` function to be echoed.

`getkey()` waits for a character to be received by the C167 serial port zero. It is common for this function to be modified to allow characters to be received from other input devices such as a keypad or alternative serial port. The source code for `_getkey()`incorporated in the Keil runtime library set can be found in "\C166\LIB\GETKEY.C".

2.3 Implementation-Specific Issues

The ISO/ANSI ‘C’ standard identifies a large number of items that are specific to the implementation of the compiler. This section seeks to characterise those areas. Where indicated, small example program has been generated to illustrate issues raised.

2.3.1.1 Test Conditions

The following compiler controls were used during testing:

WARNINGLEVEL(3)

The special WARNINGLEVEL(3) will cause the compiler to generate WARNINGs in a lint-like fashion. It will indicate most sources of potential run-time failure due to poor use of the C language or misuse of pointers. It has been used throughout the testing phase.

OPTIMIZE(X)

In some cases, the aggressive optimisation carried out by C166 prevents a proper examination of implementation-specific areas addressed in this report. In those cases where results would not be invalidated by inhibiting the optimiser, the #pragma OPTIMIZE(0) control was used.

ERRORS

Compiler errors prevent the production of an object file so therefore program cannot be linked or executed. Under such circumstances the result of the test for implementation-specific issues is irrelevant.

MOD167

The target CPU was a C167CR that contains instruction set extensions only accessible through the #pragma MOD167 control.

HLARGE

The recommended HLARGE memory model was used. A justification for this can be found in section 2.4.

2.3.2 The layout of storage for parameters.

2.3.2.1 Basic Data Types In C166

The data types available in C166 are:

```
bit                   =      1-bit          0 to 1
signed char           =      8-bits         -128 to +127
unsigned char         =      8-bits         0 to 255
signed int            =      16-bits        -32768 to -32767
signed short          =      16-bits        -32768 to -32767
unsigned int          =      16-bits        0 to 65535
unsigned short        =      16-bits        0 to 65535
signed long           =      32-bits        -2147483648 to +2147483647
unsigned long         =      32-bits        0 - 429496795
float                 =      32-bits        +/-1.176E-38 to +/-3.4E+38
double                =      64-bits        +/- 1.7E-308 to +/- 1.7E+308
pointer               =      16/32-bits     Variable address
```

Notes:

(i) *The 16-bit ANSI “`short`” type equates exactly to `int`. The latter takes the “natural” size of the CPU, here 16-bits.*

(ii) *The machine size of the C166 is 16-bits hence `int`, `short` or `unsigned int`, `unsigned short` should be used when possible to produce the most compact and efficient code. The use of `char` and `unsigned char` will result in a lot of MOVBZ-type instructions which waste time.*

(iii) *`char` and `unsigned char`: Unless a signed 8-bit number is required, always use `unsigned char`. `Char` is normally reserved for ASCII characters which have no sign.*

2.3.2.2 Representation Of Data

The C166 uses the Intel "little-endian" byte order.

2.3.2.2.1 Significant Character Values

The C166 compiler uses the following significant character values:

NEWLINE	'\n'
LINEFEED	'\r'
String null terminator	'\0'

2.3.2.2.2 Representation Of 16 Bit Quantities

Thus `int` and `short` quantities are represented as follows:

Value = 0x1234

Address 0	**Address 1**
0x34	0x12

2.3.2.2.3 Representation Of 32 Bit Quantities

`long` quantities are represented thus:

Value = 0x12345678

Address 0	**Address 1**	**Address 2**	**Address 3**
0x78	0x56	0x34	0x12

2.3.2.2.4 Representation Of Floating Point Quantities

The IEEE754 format is used and data is stored in a little-endian format

2.3.2.3 Use Of Registers And Userstack

The C166 compiler uses two areas to store local data (automatics) and function parameters. Under the default optimisation level of 6, C166 will attempt to use 12 registers in the current register bank for local data and up to 5 registers for parameter passing. Bit parameters are passed exclusively in register R15.

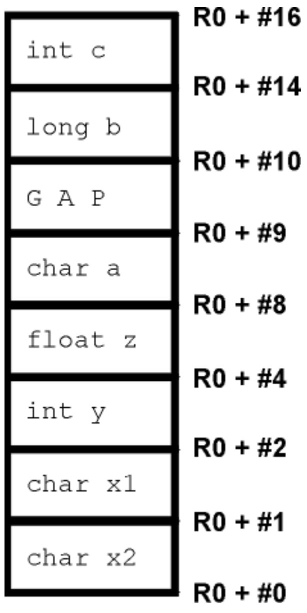
Note: The R0 register is used to create the userstack and its value **MUST NEVER BE ALTERED BY THE USER'S PROGRAM**

Example Of Userstack Layout After Function Call

Given the following function with three parameters and four local variables at OPTIMIZE level 0, the userstack will be loaded as shown in the illustration.

```
#pragma OPTIMIZE(0)

void func(char a, long b, int c) {


   char_x1 ;
   char_x2 ;
   int y ;
   float z ;

   ;
   ;

   }
```

UserStack Layout



2.3.2.4 Register Masks And Global Register Optimisation

C166 will attempt to place local data into 12 registers to improve execution speed by making best use of the single-cycle `MOV Rw,Rw` type instructions. Local data that overflows the available registers is placed on the userstack. C166 will only allocate locals to registers in a particular function provided that no further functions are called. This limitation exists as there is no way that the compiler can know whether the called function will use registers that are already used by the caller. In addition, the called function will push those registers it requires onto the system stack, with a consequent increase in run time.

The C166 compiler uses the REGISTERMASK concept to overcome this limitation. These “masks” are generated in the list file by the compiler in the form of a code string, prefixed with ‘@’, within which is contained coded information on each function’s register usage. This new information can then be automatically attached to the function prototypes to tell the compiler in advance about which registers any subsequently-called functions use and hence remove the need to push registers onto the system in called functions.

```
; FUNCTION locate_trigger_point (BEGIN  RMASK = @0x6DFF)
                                    ; SOURCE LINE # 165
00F0 F68E2C03 R    MOV     trigger_count,ZEROS
                                           ; SOURCE LINE # 166
00F4 F68E0200 R    MOV     trigger_offset,ZEROS
                                           ; SOURCE LINE # 168
00F8 E00A          MOV     R10,#00H
```

The use of REGISTERMASKs is controlled by the global **Register Optimisation** item on the *Options-Make Miscellaneous* tab in uVISION.

Note1: Although the global register optimisation mechanism has been shown to be reliable, it is recommended that it is not selected in a safety-critical system.

Note2: Although the ISO-C `register` location qualifier is compiled, it has no effect on the compiler register variable allocation strategy and is thus redundant in C166.

With assembler coded functions, the user should manually work out the register mask and add it to the assembler function's prototype at the top of the C source file.

Functions written in assembler are assumed to have the worst case register usage and so any C functions calling assembler will not have any registers available to them. It is thus up to the user to manually generate the register mask and attach it to the function prototype that the calling module sees:

```
extern void asm_func0(void) @0x0010 ;  // The assembler function uses only R4
extern void asm_func1(void) @0x0018 ;  // The assembler function uses R4 & R3
extern void asm_func2(void) @0x0000 ;  // The assembler function uses all registers
extern void asm_func3(void) @0x8000 ;  // The assembler function no registers
```

The register mask for the user's own assembler functions by reference to the following:

Bit Allocations In Register Mask

–	DPP3	MDX	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	DPP0

A one in any of the various fields in a register mask indicates the following:

MDX	=>	user's assembler function uses the multiply/divide unit
R12-R1	=>	user's assembler function uses a general purpose register
DPP3	=>	user's assembler function modifies DPP3
DPP0	=>	user's assembler function modifies DPP0

2.3.3 How a diagnostic is identified.

2.3.3.1 Error and warning messages, with example reports

A "diagnostic" is considered to be a WARNING or ERROR message. A WARINING will prevent the uVISION Make system from proceeding to the link stage. A valid OBJECT file will still be produced however. An ERROR will inhibit the production of an OBJECT file and delete any previous one that may already exist. Thus linking cannot take place.

Under the uVISION environment, WARNINGs and ERRORs are indicated by a green bar over the C line in which a problem was encountered by the compiler.

Diagnostic messages are stored in two compiler-generated output files - the .ERR and .LST files.

2.3.3.2 The C166 Compiler .ERR file

Contains a list of WARNING and ERROR messages that occurred during the compilation in a machine-readable form only. It is intended to be used only by the uViSION editor.

2.3.3.3 The C166 Compiler .LST File

The .LST file emitted by the compiler contains the original C source lines with a line number prefix and the WARNING or ERROR message placed in the file so as to indicated where the problem occurred.

Example Of A Warning Diagnostic

```
*** WARNING 34 IN LINE 7 OF MAIN.C: 'incompletevar': missing declaration specifiers
```

The WARNING number can be referenced in the Keil C166 User Manual

Example Of An Error Diagnostic

```
*** ERROR 25 IN LINE 24 OF MAIN.C: syntax error near '}'
```

The ERROR number can be referenced in the Keil C166 User Manual

Example Of A .LST File With ERROR And WARNING Messages

```
C166 COMPILER  V3.12,  MAIN
13/02/99  20:24:00  PAGE 1


DOS C166 COMPILER V3.12, COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN MAIN.OBJ
COMPILER INVOKED BY: C:\C166\BIN\C166.EXE MAIN.C DB M167 WL(3) HLARGE

stmt level    source

   1          #pragma WARNINGLEVEL(3)
   2          //#pragma OT(0)
   3          //
   4          // Sample Error And Warning Messages test
   5          //
   6          //
   7
   8          incompletevar ;   // Deliberate incomplete definition.  C166 assumes type
'int'
*** WARNING 34 IN LINE 8 OF MAIN.C: 'incompletevar': missing declaration specifiers
   9
  10          unsigned short svar ;
  11
  12          extern arr[] ;
  13
  14
  15          void main(void) {
  16   1
  17   1         arr[1] = svar ;
  18   1
  19   1         incompletevar = arr[3] ;
  20   1
  21   1         if(svar = 1) {
*** WARNING 137 IN LINE 21 OF MAIN.C: constant in condition expression
  22   2
  23   2            svar = 1   // Missing ';'
  24   2            }
*** ERROR 25 IN LINE 24 OF MAIN.C: syntax error near '}'
  25   2
  26   2         while(1) { ; }
*** ERROR 25 IN LINE 26 OF MAIN.C: syntax error near '1'
  27   2         }

C166 COMPILATION COMPLETE.  2 WARNING(S),  2 ERROR(S)
```

2.3.3.4 L166 Linker Diagnostics

The L166 linker also issues WARNING and ERROR diagnostic messages. These are stored exclusively in the .M66 file.

2.3.3.4.1 Linker Syntax ERRORs

Syntax errors in the .LIN linker control file inhibit the production of an executable OMF66 file and the .M66 map file. Details of the ERROR are reported in the .LER file. This is machine readable only and is intended only to be used by the uVISION environment.

2.3.3.4.2 Linker Linking And Location Diagnostics

WARNINGs and ERRORs messages are stored by the linker at the end of the M66 file, besides being reported in the .LER file.

Example Of WARNING

```
*** WARNING 23: NDATA/NDATA0 OR NCONST MUST FIT IN ONE 16KB PAGE
    CLASS:   NDATA0

L166 RUN COMPLETE.  1 WARNING(S),  0 ERROR(S)
```

Such a warning will still produce an executable OMF66 output file. However a run time failure may occur due to the misplacement or mismatch of data.

LINKER WARNINGS MUST NEVER BE IGNORED.

Linker ERRORs are produced by references to undefined symbols. ERROR messages are stored at the end of the .M66 file as well as being reported in the machine-readable only .LER file.   The occurance of errors during linking will prevent the production of an executable OMF66 file. Any previous OMF66 will be deleted. It is therefore not possible to execute a program containing linking errors.

Example Of ERROR

```
*** ERROR 127: UNRESOLVED EXTERNAL SYMBOL
    SYMBOL:  nonexistantfunc
    MODULE:  D:\C166DEV\ALARIS\PTRCAST\DUMMY.OBJ (DUMMY)

*** ERROR 128: REFERENCE MADE TO UNRESOLVED EXTERNAL
    SYMBOL:  nonexistantfunc
    MODULE:  D:\C166DEV\ALARIS\PTRCAST\DUMMY.OBJ (DUMMY)
    ADDRESS: 01B1H

*** ERROR 128: REFERENCE MADE TO UNRESOLVED EXTERNAL
    SYMBOL:  nonexistantfunc
    MODULE:  D:\C166DEV\ALARIS\PTRCAST\DUMMY.OBJ (DUMMY)
    ADDRESS: 01B2H

L166 RUN COMPLETE.  0 WARNING(S),  3 ERROR(S)
```

2.3.4 What constitutes an interactive device.

For the purposes of a C166 project the definition of an interactive device may sensibly restricted to the on-chip serial port or any off-chip UART that is utilised by formatted IO functions such as printf(), scanf() and their derivatives. This definition may be extended to include on- and off-chip peripherals.

2.3.5 The number of bits in a character in the execution character set.

The C166 compiler uses the ASCII 7-bit character set. Extended characters, i.e. those with a value of greater than 0x7F are not supported. Multi-byte or "wide" characters are also not supported.

2.3.5.1 Legal Characters

The available character set is summarised below:

The 26 uppercase letters of the Latin alphabet

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The 26 lowercase letters of the Latin alphabet

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

The 10 decimal digits

```
0 1 2 3 4 5 6 7 8 9
```

The following 29 graphic characters

```
! " # % & ' ( ) * + , - . / : ; < = > ? [ \ ] ^ _ { | } ~
```

plus the SPACE character, and control characters representing HORIZONTAL TAB, VERTICAL TAB and FORMFEED.

If any character not listed above is encountered in a source file, compilation ends and an ERROR message generated.

2.3.5.2 Termination Of Character Strings

Strings of characters in the source character set are terminated with a character having numerical value ZERO.

2.3.6 The result of casting a pointer to an integer or vice versa.

Note: This aspect of the C166 compiler implementation is very much dependent on the model used.

Warning: This will result in unpredictable accesses to memory if a pointer is used that was the result of a cast from `int`. **THIS MUST NOT BE DONE.**

Only `unsigned int` types can be cast to pointers but even then the user must take into account the effect of the data page pointer mechanism under all memory models except HCOMPACT and HLARGE. Such programming techniques are not advisable in any commercial application.

It should be noted that all pointers that have no explicitly stated memory space qualifier under the HLARGE and HCOMPACT models contain 4 bytes and are thus more correctly converted to `long` types.

Cast `int` type to pointer

If the value of the int is < 0x8000 then an address will be access as expected under the HCOMPACT and HLARGE models. Under all other models unexpected results may be obtained.

Cast pointer to `int`

Under HLARGE and HCOMPACT models the address contained in the pointer is transferred to the `int` as expected provided it is < 0x10000.

Note: **It would be more appropriate to cast the pointer to an object of type `long`.**

Refer to the section on recommend memory models for further information

A full review of casting basic types to pointers and the associated risks is given in Appendix B

2.3.7 What constitutes an access to an object that has volatile-qualified type.

```
short volatile *xptr  ;
```

A `volatile` quantity is one whose value may change without CPU and hence compiler intervention. Memory-mapped IO devices and C167CR sfrs are regarded as `volatile`. Pointers used to test RAM areas must also be `volatile.`

2.3.8 The maximum number of declarators that may modify an arithmetic, structure or union type.

This item has been referred to KEIL

2.3.9 Whether the value of a single-character character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Whether such a character constant may have a negative value.

The C166 compiler has common source and execution character sets. Thus a single-character character constant in a constant expression that controls conditional inclusion will always match the value of the same character constant in the execution character.

It should be noted that character constants used in conditional include controls must not be simple numerical quantities, either positive or negative. Furthermore, all conditional inclusion constants, manifest constants and macros must not begin with a numerical character of any kind.

2.3.10 The mapping of source file character sequences

The C166 compiler uses identical source and execution character sets. There is therefore no mapping of characters between source and execution sets.

2.3.11 The null pointer constant to which the macro NULL expands.

The NULL macro expands to (void *) 0L) . A NULL (or uninitialised) pointer will return the value found at location 0x0000. In most cases this will be byte value 0xFA, this being the first byte of the JMPS instruction that is normally located at address 0x0000.

Example:

```
int *test_ptr ;  // Pointer will point at NULL and return int value at address 0x0000
```

2.3.12 CTYPE.H Functions

Note1: All ctype.h functions return value 0x01 for true and value 0x00 for false. Such functions operate on and return *`int`* values.

Note 2: All ctype.h functions return false (0x00) when a parameter is passed that is not part of the range of characters that it is designed to check for. All such functions behave deterministically at all times.

2.3.12.1 The Sets Of Characters Tested For By *isalnum*,

`isalnum` tests the input parameter for alphanumeric characters in the ranges 'A' to 'Z', 'a' to 'z' and '0' to '9'.

2.3.12.2 The Sets Of Characters Tested For By *iscntrl*,

The `iscntrl()` function tests for any control character whose value lies from 0x00 (NUL) through 0x1F (US) or has a value of 0x7F.

2.3.12.3 The Sets Of Characters Tested For By *islower*

The islower() function tests for any character that is a lowercase letter in the range of 'a' to 'z'.

2.3.12.4 The Sets Of Characters Tested For By *isprint*

The `isprint()` function tests for any printing character in the range 0x20 to 0x7E.

2.3.12.5 The Sets Of Characters Tested For By *isupper*

The isupper() function tests for any character that is a uppercase letter in the range of 'A' to 'Z'.

2.3.13 The values returned by the mathematics functions on domain errors.

Only those mathematical functions that can generate domain or range errors are listed here. Their behaviour under such conditions is documented.

2.3.13.1 C166 Mathematical Function List

Note**:** the double type == float unless *`#pragma FLOAT64`* is in force.

```
int     abs  (int   val);
long   labs  (long  val);
double fabs  (double val);
double sqrt  (double val);
double exp   (double val);
double log   (double val);
double log10 (double val);
double sin   (double val);
double cos   (double val);
double tan   (double val);
double asin  (double val);
double acos  (double val);
double atan  (double val);
double sinh  (double val);
double cosh  (double val);
double tanh  (double val);
double atan2 (double y, double x);

double ceil  (double val);
double floor (double val);
double modf  (double val, double *n);
double pow   (double x, double y);

unsigned int _chkfloat_  (float x);
unsigned int _chkdouble_ (double x);
```

2.3.13.2 Floating Point Limits

Single Precision

Not a number (NaN) for float types = 0xFFFFFFF

Positive Infinity = +INF = 0x7F80000

Negative Infinity = -INF = 0xFF80000

Double Precision

Not a number (NaN) for double types = 0xFFFFFFFFFFFFFFFF

Positive Infinity = +INF = 0x7FF0000000000000

Negative Infinity = -INF = 0xFFF0000000000000

Note: there is no provision for floating point exceptions in C166 and it is up to the user to adequately respond to floating point errors.

2.3.13.3 Trigonometric Functions That Are Subject To Domain Or Range Errors

2.3.13.4 The acos Function

Synopsis

```
#include <math.h>
double acos(double x);
```

Description

The acos function computes the principal value of the arc cosine of x. A domain error occurs for arguments not in the range [-1, +1].

Returns

The acos function returns the arc cosine in the range [0, pi] radians. The domain error value returned is NaN (0xFFFFFFFF)

2.3.13.4.1 The asin Function

Synopsis

```
#include <math.h>
double asin(double x);
```

Description

The asin function computes the principal value of the arc sine of x. A domain error occurs for arguments not in the range [-1, +1].

Returns

The asin function returns the arc sine in the range +/-pi/2 radians. The domain error value returned is NaN (0xFFFFFFFF)

2.3.13.4.2 The atan2 Function

Synopsis

```
#include <math.h>
double atan2(double y, double x);
```

Description

The atan2 function computes the principal value of the arc tangent of y / x, using the signs of both arguments to determine the quadrant of the return value. A domain error may occur if both arguments are zero.

Returns

The atan2 function returns the arc tangent of y / x, in the range [-pi, +pi] radians. No domain error value is return if input parameters are both zero.

Caution: The user should check for zero parameters before calling the function

2.3.13.5 Hyperbolic Functions That Are Subject To Domain Or Range Errors

2.3.13.5.1 The cosh Function

Synopsis

```
#include <math.h>
double cosh(double x);
```

Description

The cosh function computes the hyperbolic cosine of x. A range error occurs if the magnitude of x is too large.

Returns

The cosh function returns the hyperbolic cosine value. The range error value returned is INF (0x7F8000)

2.3.13.5.2 The sinh Function

Synopsis

```
#include <math.h>
double sinh(double x);
```

Description

The sinh function computes the hyperbolic sine of x. A range error occurs if the magnitude of x is too large.

Returns

The sinh function returns the hyperbolic sine value. The range error value returned is INF (0x7F8000)

2.3.13.6 Integer Arithmetic Functions

2.3.13.6.1 The abs function

Synopsis

```
#include <stdlib.h>
int abs(int j);
```

Description

The `abs` function computes the absolute value of an integer `j`. If the parameter is the most negative integer, the behaviour is undefined.

Returns

The `abs` function returns the absolute value. If the input parameter is the most negative integer, the result is the most negative integer, no action having been taken by `abs`().

2.3.13.6.2 The labs function

Synopsis

```
#include <stdlib.h>
long labs(long  j);
```

Description

The `labs` function is equivalent to the `abs` function except that the argument and the return value each have type `long`.

Returns

The `labs` function returns the absolute value. If the input parameter is the most negative integer, the result is the most negative integer, no action having been taken by `labs()`.

2.3.14 Compiler Actions With Unusually Terminated Source Files

This section determines the outcome when a non-empty source file is compiled and:

2.3.14.1 Source file does not end in a new-line character

Result: Compilation succeeds without error or warning

2.3.14.2 Source file ends in a new-line character immediately preceded by a backslash character

Result:

The compiler issues ERROR 25; description: syntax error near '<EOF>'

Example

```
C166 COMPILER  V3.12,  MAIN1
13/02/99  18:43:04  PAGE 1


DOS C166 COMPILER V3.12, COMPILATION OF MODULE MAIN1
OBJECT MODULE PLACED IN MAIN1.OBJ
COMPILER INVOKED BY: C:\C166\BIN\C166.EXE MAIN1.C WL(3)

stmt level    source

   1          #pragma OT(0) WL(3)
   2          //
   3          // End of file is backslash newline
   4          //
   5
   6          int test0 ;
   7          int test1 ;
   8
   9          void main(void) {
  10   1
  11   1         test0 = 1 ;
  12   1         test1 = 2 ;
  13   1
  14   1         }\
*** ERROR 25 IN LINE 14 OF MAIN1.C: syntax error near '<EOF>'
*** ERROR 7 IN LINE 14 OF MAIN1.C: compilation aborted

C166 COMPILATION COMPLETE.  0 WARNING(S),  2 ERROR(S)
```

2.3.14.3 Source file ends in a partial pre-processing token or comment.

Result:

The compiler issues ERROR 300; description: unterminated comment

Example

```
C166 COMPILER  V3.12,  MAIN3
13/02/99  18:44:17  PAGE 1


DOS C166 COMPILER V3.12, COMPILATION OF MODULE MAIN3
OBJECT MODULE PLACED IN MAIN3.OBJ
COMPILER INVOKED BY: C:\C166\BIN\C166.EXE MAIN3.C WL(3)

stmt level    source

   1          #pragma OT(0) WL(3)
   2          //
   3          // End of file is in comment
   4          //
   5
   6          int test0 ;
   7          int test1 ;
   8
   9          void main(void) {
  10   1
  11   1         test0 = 1 ;
  12   1         test1 = 2 ;
  13   1
  14   1         }
  15
  16          /* This is a test comment
*** ERROR 300 IN LINE 16 OF MAIN3.C: unterminated comment

C166 COMPILATION COMPLETE.  0 WARNING(S),  1 ERROR(S)
```

2.3.15 The outcome when the result of an integer arithmetic function (*abs*, *div*, *labs*, or *ldiv*) cannot be represented.

div and *ldiv* are not implemented in C166. a*bs* and *labs* do not return unrepresentable values. In the cases where the smallest possible numbers (-32768 and - 2147483648 respectively) are passed, the functions do not attempt to produce the absolute value and simply return the input parameter unchanged.

2.3.16 The outcome when an *lvalue* with an incomplete type is used in a context that requires the value of the designated object.

Incomplete types may be declared and compiled successfully. However any attempt to use an object derived from an incomplete type will result in an error message.

Example

```
struct test1 ;  // No member list
struct test1 test2, test3 ; // Create two identical structures


void main(void) {


   test3 = test2 ;  // Attempt to copy structure fails
```

Result:

```
*** ERROR 129 IN LINE 45 OF MAIN.C: 'test3' uses undefined struct/union 'test1'
*** ERROR 129 IN LINE 45 OF MAIN.C: 'test2' uses undefined struct/union 'test1'
*** ERROR 52 IN LINE 45 OF MAIN.C: use of undefined type 'test1'
```

2.3.17 Mismatches in type between *lvalue* and object

The outcome when an object has its stored value accessed by an *lvalue* that does not have one of the following types:

- the declared type of the object
- a qualified type of the declared type of the object
- the signed or unsigned type corresponding to the declared type of the object
- the signed or unsigned type corresponding to a qualified version of the declared type of the object
- an aggregate or union type that (recursively) includes one of the aforementioned types among its members
- a character type.

2.3.17.1 Summary Of Results Under Type Mismatch Conditions

Conversion from `bit` to `int`, `long`, `float`, `double` is carried out with no loss of data

Conversion from `int` to `long`, `float`, `double` is carried out with no loss of data

Conversion from `float` to `double` is not required as both `float` and `double` are 64 bits when #pragma FLOAT64 is in force.

Conversion from any other type to `bit` results in one if the source object is non-zero. It otherwise results in zero.

Conversion to `int` from `float` is only valid if value of source object is < 32768.00000 (2P15)

Conversion to `int` from `double` under FLOAT64 is only valid if source object is < 32768.00000

Conversion to `long` from `float` is only valid if source object is < 2147483648.00000 (2P31)

Conversion to `long` from `double` under FLOAT64 is only valid if source object is < 2147483648.00000 (2P31)

Note: When converting from float types to integer or long, care must be taken that the value is not greater than the maximum value that the target object can contain.

Further information on conversions between integer and float types is given in section 2.3.23 to 2.3.24.3

2.3.18 Incomplete Data Declarations

The outcome when an identifier for an object is declared with no linkage and the type of the object is incomplete after its declarator, or after its init-declarator if it has an initializer.

Example:

```
int complete, idata incomplete_idata, near incomplete_near, arr[]  ;
//
// int in NDATA, int in IDATA, int in NDATA, object not created
//
int x, idata y, near z,  arr_init[] = { 1,2,3,4 }  ;
//
// int in NDATA, int in NDATA, int array in NDATA with initialised data
//
```

Remarks

Objects declared with an incomplete type take the default type of "`int`".

Individual objects in lists of object declarations do not assume the type or location qualifier of the preceding object.

Example:

```
int a, idata b, near c ;
//
// int in NDATA, int in IDATA, int in NDATA
//
```

The `idata` qualifier for object b is not applied to `c`, or any other object declaration that might follow in the list.

Note: At WARNINGLEVEL(2) or above an incomplete declaration will result in a WARNING 34. The object created will be of type `'int'`

Example:

```
34          incompletevar ;
*** WARNING 34 IN LINE 34 OF MAIN.C: 'incompletevar': missing declaration specifiers
```

2.3.19 Volatile/non-volatile mismatches

The outcome when an attempt is made to refer to an object with volatile-qualified type by means of an *lvalue* with non-volatile qualified type.

Example

```
   testvar = exvar ;
   testvar = exvar ;
   testvar = exvar ;
   testvar = exvar ;
   testvar = exvar ;
   testvar = exvar ;
   testvar = exvar ;
```

Data is transferred to *lvalue* at each attempted access to the `volatile` type.

Example Of Sequential Volatile Accesses

```
/+   CSP:0x0122     E014     MOV    R4,#1
/+   CSP:0x0124     F6F40090 MOV    exvar,R4
/+   #14      testvar = exvar ;
/+   CSP:0x0128     F2F40090 MOV    R4,exvar
/+   CSP:0x012C     F6F40290 MOV    testvar,R4
/+   #16      testvar = exvar ;
/+   CSP:0x0130     F2F40090 MOV    R4,exvar
/+   CSP:0x0134     F6F40290 MOV    testvar,R4
/+   #18      testvar = exvar ;
/+   CSP:0x0138     F2F40090 MOV    R4,exvar
/+   CSP:0x013C     F6F40290 MOV    testvar,R4
/+   #20      testvar = exvar ;
/+   CSP:0x0140     F2F40090 MOV    R4,exvar
/+   CSP:0x0144     F6F40290 MOV    testvar,R4
/+   #22      testvar = exvar ;
/+   CSP:0x0148     F2F40090 MOV    R4,exvar
/+   CSP:0x014C     F6F40290 MOV    testvar,R4
/+   #24      testvar = exvar ;
/+   CSP:0x0150     F2F40090 MOV    R4,exvar
/+   CSP:0x0154     F6F40290 MOV    testvar,R4
/+   #26      testvar = exvar ;
/+   CSP:0x0158     F2F40090 MOV    R4,exvar
/+   CSP:0x015C     F6F40290 MOV    testvar,R4
/+   #28      while(1) { ; }
/+   CSP:0x0160     0DFF     JMPR   CC_UC,^#28
/+   ?C_ENDINIT     0000     ADD    R0,R0
```

2.3.20 Incomplete Types And Tentative Declarations

The outcome when an identifier for an object with internal linkage and an incomplete type is declared with a tentative definition.

The declaration:

```
static short arr[] ;
```

will result in an error and compilations ends. No object file is produced and the build process stops. There is thus no hazard resulting from this tentative declaration.

2.3.21 Shift Left And Shift Right Operations

The implementation of the 'shift left' and 'shift right' operators shall be documented.

2.3.21.1 Left Shift - Signed

```
signedtest = 0x0001 ;

   for(i = 0 ; i < 16 ; i++) {
      signedtest <<= 1 ;
      }
```

The object is shifted left with zero shifted into the least significant bit. The most significant bit is shifted into the carry flag of the C167 program status word (PSW)

2.3.21.2 Right Shift - Signed

```
signedtest = 0x8000 ;

for(i = 0 ; i < 16 ; i++) {
   signedtest >>= 1 ;
   }
```

If the most significant bit (b15) was set, ONES are shifted into most significant bit.

If the most significant bit (b15) was clear, ZEROS are shifted into most significant bit.

The least significant bit is shifted into carry flag

2.3.21.3 Left Shift - Unsigned

```
unsignedtest = 0x0001 ;

for(i = 0 ; i < 16 ; i++) {
   unsignedtest <<= 1 ;
   }
```

The object is shifted left with zeros shifted into the least significant bit. The most significant bit is shifted into the carry flag of the C167 program status word (PSW)

2.3.21.4 Right Shift - Unsigned

```
unsignedtest = 0x8000 ;

for(i = 0 ; i < 16 ; i++) {
   unsignedtest >>= 1 ;
   }
```

The object is shifted right with zeros shifted into the most significant bit. The least significant bit is shifted into the carry flag of the C167 program status word (PSW)

2.3.22 Integer Division Behaviour

The implementation of integer division shall be determined, documented and taken into account. Attention shall be paid to the sign of the remainder.

Positive dividend & divisor => Positive result, positive remainder

Positive dividend, Negative divisor => Negative result, positive remainder

Negative dividend, positive divisor => Negative result, negative remainder

Negative dividend, negative divisor => Positive result, negative remainder

Positive dividend, zero divisor => indeterminate result

Negative dividend, zero divisor => indeterminate result

2.3.23 Floating Point Truncation

The manner in which the chosen compiler implements truncation and rounding when a floating point number is converted to a smaller floating point number shall be determined, documented and taken into account.

The Keil C166 supports single precision `float` and double precision `double` floating point types to the IEEE754 standard. Without the #pragma FLOAT64, the `double` type is implemented as `float`. With #pragma FLOAT64, `float` and `double` are implemented as `double.` It is therefore not possible to convert a double precision `double` to `float.` There is therefore no potential for loss of data through truncation.

2.3.24 Float To Integer Conversions

The implementation of truncation from floating point quantities to integer types in the chosen compiler shall be determined, documented and taken into account.

Converting from floating point types to integer types is not recommended. It should only be done once and only then during some form of user interface function. Floating point numbers are only an approximation to true values and the behaviour of conversions of float values near to integer values is not defined. It is thus difficult to quantify the exact outcome of conversions and care must be exercised.

Floating point types must not be used in any control loop or function, being reserved only for user interface functions.

Some general guidelines can be given however:

If a floating point object has a positive or negative value greater than the highest value of an integer type but less than the highest value of a long type, the value transferred is meaningless.

Caution: Before converting any float to an integer type, the value must be checked for potential overflow which would give a meaningless result.

If a floating point object has a positive or negative value greater than the highest value of a `long` type the value transferred is 0xFFFFFFF. This value will also be transferred to an integer type.

Caution: Before converting any float to a long type, the value must be checked for potential overflow and hence a meaningless result.

2.3.24.1 Single Precision

Float (single precision) represents 6 to 7 significant digits. However it has been determined that in typical engineering calculations only 5 digits can be relied on.

```
var32 = 65535.99 ;
```

If the length of the initialisation value exceeds 7 significant digits the entire fractional portion is lost.

Rounding up of fractional parts only occurs when the value to be converted is within one unit of resolution i.e. the lowest representable value possible given the precision and size of the value to be converted. The absolute value of this lowest representable value is dependent entirely on the magnitude of the object and thus cannot be defined here.

Example:

float **integer**

65535.99 = 65535 = 0xFFFF

65535.99 + 00000.01 = 65536.00 = 0x10000

2.3.24.2 Double Precision

Double represents 14-15 significant digits. However it has been determined that in typical engineering calculations only 13 digits can be relied on.

```
var64 = 65535.5000000001 ;
```

If the length of the initialisation value exceeds 14 significant digits the entire fractional portion is lost.

Rounding up of fractional parts only occurs when the value to be converted is within one unit of resolution i.e. the lowest representable value possible given the precision and size of the value to be converted. The absolute value of this lowest representable value is dependent entirely on the magnitude of the object and thus cannot be defined here.

Example

double	**integer**
65535.9999999999	= 65535 = 0xFFFF
65535.9999999999	+ 00000.0000000001 = 65536 = 0x10000

2.3.24.3 Limits Of Single And Double Precision Floating Point Types

The C166 compiler has the following maximum and minimum values for floating point quantities:

Maximum float value	3.40282e38
Minimum float value	1.17549e-38
Maximum double value	1.79769313486231500e+308
Minimum double value	2.22507385850720200e-308

Note: Single precision floating point numbers are stored in the following arrangement:

Address:	**+0**	**+1**	**+2**	**+3**
Contents:	MMMM MMMM	MMMM MMMM	MMMM MMMM	SEEE EEEE

The mantissa "MMM....MMMMM" is a 24 bit quantity. The exponent "EEE EEEE" is stored as a twos complement value with a 127 offset. The sign of the exponent "S" is a single bit value.

2.3.25 Type promotion, `char` To `int`

Automatic promotion of `char` to `int` only occurs in arithmetic operations. Therefore no explicit casting from `char` to `int` is required in the following example:

Example:

```
resultchar0 = (varchar0 * varchar1)/varchar2 ;
```

Gives the correct result as the implied 16 bit value in the numerator is allowed for by the use of a 16x8 divide instruction.

***Note*:** Promotion from char to int in compare operations does **not** occur.

2.3.26 Type Promotion From int To `long`

As dictated by the ISO C standard, there is no automatic promotion from `int` to `long`. This must be considered in the following situation.

```
unsigned int x, y,z ;

z = (x * y) / z ;
```

Here there is an implied `long` value in the numerator that will be lost unless the appropriate cast to `long` is used. The correct ISO C way to perform this calculation is:

```
unsigned int x,y,z ;

z = ((unsigned long)x * (unsigned long)y)/(unsigned long)z ;
```

In tested version of C166, the following efficient code will result:

```
;    a = ((long)x * (long)y)/(long)z ;
                                               ; SOURCE LINE # 20
      MOV     R5,WORD y
      MOV     R4,WORD x
      MULU    R4,R5
      MOV     R6,WORD z
      DIVLU   R6
      MOV     R4,MDL
      MOV     WORD a,R4
```

2.4 Compiler System Controls Which Impact Software Integrity And Maintainability

2.4.1 Overview

This section is the most important part of the report when assessing the integrity of the C166 executable within the product Assuming that the software has been written in accordance with the standards and recommendations listed in the project coding standard, the highest risk of malfunction will be due to CPU configuration errors and incorrect placement of specific ROM and RAM objects.   The sizing and placement of SYSTEM and USER stack must also be verified.

A detailed examination of the START167.A66 assembler file should be made to ensure that the CPU bus interface is configured in accordance with the hardware design parameters. This topic is outside the scope of this document.

The L166 linker control file should be assessed to ensure that ROM and RAM objects are located at valid hardware addresses. Moreover the limitations on physical placement of certain CLASSES must be borne in mind. This topic is considered to be within the scope of this report and is examined in section 2.4.4.3 and 2.4.4.8

Other C167-specific safety-critical items to be checked for:

- Critical interrupt regions and atomic sequences
- Read-modify-write pipeline effects
- Data coherency in 32-bit quantities updated by interrupts
- Illegal sharing of interrupt priority and group level
- Illegal sharing of registerbanks by interrupt functions
- Misuse of near and huge pointers
- Trapping of runtime exceptions via the CLASS A & B trap system.
- Unexpected results with pointer arrays to constant strings

Note: To satisfactorily audit these aspects, specialist knowledge of the C167 architecture and C166 compiler/linker relationship is required.

2.4.2 Compiler Controls

The objectives of portability, ease of programming and most efficient use of the C167 architecture can be attained through the correct choice of memory model. For all external and internal ROM C167 systems the HLARGE memory model provides the best fit to these objectives.

Under this model the user need only use the "`idata`" keyword to place data into the on-chip RAM area of the C167 CPU and "`sdata`" to place data in the XRAM area(s).

The user should be fully acquainted with the consequences of using the following controls if they are applied or check whether the programming techniques employed require their use.

NODPPSAVE

On the C167, DPP3 is always set to 3 to indicate the base of the SYSTEM area at 0xC000 and DPP0 is never changed from its power-up value, unless the user alters it himself. Therefore, C166 need not stack these two registers on entry to an interrupt routine, thereby saving two PUSHes and POPs (0.4us @20MHz).

NOALIAS

The NOALIAS control can cause unexpected results in when global or static objects are modified by pointers. In most cases source code that could be upset by this control does not conform to the MISRA-C guidelines and should therefore not be present anyway. It is recommended that this control is not used.

Example Taken From C166 User Manual:

Without NOALIAS

```
struct { int i1; inti2; } *p_struct ;
int val ;

void func1(int *p_val) {
   p_struct->i1 = val ;    // Read val
   *p_val = 0 ;            // Zero val via pointer
   p_struct->i2 = val ;    // Read val again.  Now it is zero
   }


void func2(void) {
   func1(&val) ;
   }
```

With NOALIAS

```
struct { int i1; inti2; } *p_struct ;
int val ;

void func1(int *p_val) {
   p_struct->i1 = val ;    // Read val
   *p_val = 0 ;            // Zero val via pointer
   p_struct->i2 = val ;    // Read val again.  Value from first read carried forward
   }                       // so zeroing by pointer in previous line is not seen


void func2(void) {
   func1(&val) ;
   }
```

NOFIXDPP

The user stack is by default in the near data area which is addressed by DPP2. The NOFIXDPP control is required if the DPPUSE control is used at the linking stage. It is also used if the USER STACK is moved to the on-chip IDATA RAM, as required in many single chip applications.

STATIC

Normally, C166 will try to put as many local variables into registers (R1-R15) as possible as the MOV Rw,Rw register-to-register instructions execute in 100ns (@20MHz). All the normal ADD, SUB, CMP type instructions are available in the register-to-register variety so that any such operation will 100ns at 20MHz. Variables that overflow the available local registers are placed on the “User Stack”, as in a PC-type compiler and are addressed via MOV R1,[R0 + #displacement] type instructions. As a RISC CPU, there are very few “stack-relative” instructions so that operating on user stack variables usually takes several instructions.

A significant performance advantage for interrupt functions or those with a large number of local variables can therefore be had by forcing the compiler to put locals that cannot fit into registers (R1-R15) into (near) static RAM segments to create a “compiled” stack, as in the C51 compiler. The common ADD, SUB and CMP instructions all can operate directly on RAM so that there is little performance loss when compared to register variables.

Note: Any functions within modules compiled with this control will no longer be reentrant, thus if enabled here in the C166 Options menu, no reentrancy will be possible across the entire program which may not be acceptable. This control is therefore better used as a #pragma STATIC with only those modules which contain functions which are used non-reentrantly, such as interrupt routines.

HOLD(location,size)

Instructs the compiler to put any object less than the given size into the stated memory area. In the example, any object of 4 bytes or less (i.e. long, short, int, char) should be placed on-chip. This is a way of making sure that large objects such as arrays and structures do not eat up valuable fast on-chip RAM. Here, IDATA means the on-chip RAM at 0xFA00 (0xF600 on the C167).

PECDEF(a,b,c,...)

To ensure that the linker does not attempt to place data at the PEC pointer addresses, the user must list the PEC channels initialised as arguments to the PECDEF control Failure to do so will result in malfunctions of the PEC system with unpredictable results.

2.4.3 Linker Controls

The use of the linker via its .LIN control file is central to ensuring correct and reliable system operation. The user should satisfy himself that the addresses give for the CLASSES and SECTIONS created by the compiler are suitable for the target hardware platform. Some guidance on this is given in section 2.4.3.

2.4.4 CPU Configuration Controls In START167.A66

The proper configuration of the START167.A66 file is crucial to correct system performance. The user should verify that the settings made are appropriate to the hardware design of the target system. This topic is outside the scope of this report but Hitex would be willing to assist in any appraisal of the configuration proposed or in use.

2.4.5 Characteristics Of HLARGE Model Programs

2.4.5.1 Defaults Under HLARGE Model

Default data memory space for HLARGE model:

```
huge
```

Under HLARGE, any data declaration that does not contain a location qualifier will be of type huge except under circumstances described in section 2.4.1.3 under the #pragma HOLD() control.

Examples

```
unsigned short hvar ;    // Variable in HDATA0 class
unsigned short *hptr ;   // Pointer to huge (default) unsigned short object
unsigned short const hconstant = 2 ; // unsigned short constant in HCONST class in ROM area
```

2.4.5.2 Data Placement And Size Limitations

- Total size of all data declared without `huge` memory space qualifier <= 16MB
- Largest single array, structure or union <= 0xFFFF bytes
- No array, structure or union must be allowed to straddle a 64k segment boundary.

  Note: The L166 linker will not permit this to happen
- No pointer must be incremented or decremented across a 64k segment boundary

  Note: Pointers used for RAM tests, checksum or CRC must be of type `xhuge`.
- Pointers occupy 4 bytes
- Executable functions may be located at any memory address.

2.4.5.3 HLARGE Model Data CLASS Placement

HDATA, HDATA0, HCONST classes ideally should be located on 64k segment boundaries.

NDATA, NDATA0 classes must be located on a page boundary, i.e. an address divisible by 0x4000.

IDATA and IDATA0 classes must be located at 0xF600

SDATA and SDATA0 classes must be located in the region 0xC000 - 0xFDFF

FCODE class must be locate in the ROM area

NCONST class must be locate on a 0x4000 byte boundary in the ROM area

The sections "?C_CLRMEMSEC" and "?C_INITSEC" must be located near the base of he FCODE class in ROM.

2.4.5.4 Management Of Near Data (NDATA) Under HLARGE

The #pragma HOLD(near,6) control is applied by default. This has the effect of placing all objects of up and including 6 bytes in length into the near data classes. Thus all `char`, `int`, `short` and `long` types will be placed into a 16k "`near`" block (NDATA & NDATA0 classes) that has the benefit of faster addressing. Small arrays of up to 6 bytes will also be placed into the `near` area. The total of all such `near` objects must not exceed 16kb.

The NDATA and NDATA0 classes must be placed at 0x4000 byte boundaries. The end address of these classes must not be more than 0x4000 bytes above the start address and must be stated.

2.4.5.5 High Speed Data

Variable data accessed frequently can benefit by being located in the C167 on-chip RAM. This can be achieved via the `idata` location qualifier.

Example Of idata

```
unsigned short idata fast_variable ; // Put this data on-chip
```

2.4.5.6 Very Large Objects

Data objects of above 0xFFFF bytes in size should be declared with the xhuge keyword. Such objects can be up to 16MB in size.

Example

```
unsigned short xhuge verybigarray[0x20000] ; // A very big array
unsigned short xhuge *ram_memtestptr ;       // A pointer than will test 256kb RAM
```

Objects declared with `xhuge` will be placed into the XDATA and XDATA0 classes. These can be placed at any address.

2.4.5.7 Typical Pointer Declarations

A generalised pointer declaration is thus:

```
<type> <const/volatile> <typequalifier> * <dataname> = (<type> <const/volatile>) <address>;
```

Pointers defined with nothing between the asterisk '*' and the `<dataname>` are located by default in the data class determined by the memory model. It is entirely possible to force the pointer itself into a specific memory space. By putting a type and `typequalifier` after the '`*`', the pointer can be placed. This is most frequently required when is desired to put a pointer into EPROM, i.e. a constant class. Jump tables or tables of function pointers fall into this category also.

Example - Put a huge constant pointer to a constant string into EPROM itself

```
char const huge * const fixed_string[] = { "String in EPROM " } ;
```

Example - Put a huge constant pointer to a ram variable into EPROM

```
char huge ram_variable = 1 ;  // Variable in far RAM

char huge * const fixed_string[] = &ram_variable ;  // Set up a pointer in EPROM to ram
variable
```

Example - Create a pointer suitable for performing a destructive read/write RAM test over a 256k RAM that is not itself within the RAM to be tested. The pointer to be located in the on-chip RAM

```
unsigned short xhuge volatile * idata mem_ptr ;
```

2.4.5.8 Typical Linker Control File

```
// This is intended as an example only and users must satisfy themselves as to
// suitability of this file to their application.
//
// Fix classes for HLARGE model
//
CLASSES(ICODE(000200H),
        FCODE(000200H-0007FFFH,018000H-02FFFFH),
        NCONST(004000H-07FFFH),
        IDATA(00F600H-00FDFFH),
        IDATA0(00F600H-00FDFFH),
        FCONST(018000H),
        HCONST(018000H),
        SDATA(00C000H-00DFFFH),
        SDATA0(00C000H-00DFFFH),
        NDATA(040000H-043FFFH),
        NDATA(040000H-043FFFH),
        FDATA(040000H),
        FDATA0(040000H),
        HDATA(040000H),
        HDATA0(040000H),
        XDATA0(040000H),
        XDATA(040000H))
//
//
SECTIONS(?C_CLRMEMSEC(0x400),  // Put initialisation tables into ROM
       ?C_INITSEC)
//
// End of linking process....
//
```

2.4.6 The Special Sections "?C_CLRMEMSEC" And "?C_INITSEC".

These are special sections created by C166 to hold the initial values of RAM variables. When a variable is declared such as `int x = 0x80`, the 0x80 is actually placed in ROM-based look up table. During the startup.a66, this ROM data is transferred to its final resting place in RAM. By default L166 will place these near the bottom of memory, on the assumption that this must be EPROM. On systems where the program is at, for example 0x40000, it is up to the user to put these into the correct area.

This is achieved by using the sections control in L166. They are best kept together; the only point to watch is that the size of the two sections will vary according to how much initialised data there is in the program. Thus, it is quite possible that they will grow such that they will overlap or move into an illegal memory area. The linker will issue a warning which must not be ignored.

Example:

```
main.obj,&
start167.obj &
to exec &
VECTAB(0x40000) &
CLASSES(NCODE(0x40400),NCONST(0x40400),NDATA(0x48000)) &
SECTIONS(?C_CLRMEMSEC(0x40400),?C_INITSEC)
REGFILE(exec.reg)
```

2.5 Floating Point Library Test

A *paranoia* check will be performed to validate the adequacy of the floating point arithmetic provided by the compiler.

2.5.1 Preliminary Work

The supplied PARANOIA program was compiled with the Keil C166 compiler. Some modifications were required to the keyboard reading function to allow the required user input. The HLARGE memory model was used and the single floating point (default) mechanism enabled.

2.5.2 Preliminary Results

It was noted that the program source was seriously at variance with the requirements of MISRA-C and accepted good C programming practice. A large number of `setjmp`, `goto` and other inappropriate statements were observed. The operation of the program was undocumented.

The program consisted of one enormous monolithic function and if nothing else, the ability of C166 to compile it without error or memory exhaustion provides considerable reassurance as to the compiler's robustness. The suitability of the program for testing floating point performance of an embedded CPU is questionable.

As a confidence check, the program was also tested on the alternative Tasking C166 compiler and "industry standard" Microsoft C v11.00 compiler. The full report listings are given in Appendix E.

2.5.2.1 Keil C166 3.12j Results

```
The number of  FAILUREs  encountered =       3.
The number of  SERIOUS DEFECTs  discovered = 1.
The number of  DEFECTs  discovered =         3.
The number of  FLAWs  discovered =           1.

The arithmetic diagnosed has unacceptable Serious Defects.
Potentially fatal FAILURE may have spoiled this program's subsequent diagnoses.
END OF TEST.
```

2.5.2.2 Tasking C166 v6r3 Results

Program failed to compile and no executable could be produced. This has been referred to Tasking for comments.

2.5.2.3 Microsoft C v11.00 Results

Note: To force the compiler to behave in an equivalent mode to the Keil C166, the 386 code generator was used and the floating point emulator library used. These measures prevented the Pentium floating point co-processor being used.

```
The number of  FAILUREs  encountered =       6.
The number of  SERIOUS DEFECTs  discovered = 5.
The number of  DEFECTs  discovered =         10.
The number of  FLAWs  discovered =           2.

The arithmetic diagnosed has unacceptable Serious Defects.
Potentially fatal FAILURE may have spoiled this program's subsequent diagnoses.
END OF TEST.
```

2.5.3 Discussion Of Results

The Keil compiler was the only one tested that was able to compile and run the test program without major modification. The results appear to suggest that the floating point performance is inadequate but this should be viewed in the context that the notionally equivalent Tasking C166 compiler was unable to compile the test piece and that the reported performance of an "industry standard" PC compiler was significantly worse.

Note: The user should examine the results presented here and in Appendx E. In the light of these and on the actual usage of floating point quantities in the application, the user should decide what action to take.

2.6 Limits Of The Keil C166 Compiler System

2.6.1 Compiler Implementation Limits

1. Maximum of 20 levels of indirection to any standard data type.
2. Names can be up to 255 characters long. Only the first 32 are significant. Names are case-sensitive.
3. The maximum number of cases in a `switch` block is not fixed and is limited only by the available memory size and the maximum size of individual functions.
4. The maximum number of nested function calls in and invocation parameter list is 10.
5. The maximum number of nested include files is 9.
6. The maximum depth of directives for conditional compilation is 20.
7. Instructions blocks ({...}) may be nested up to 32 levels.
8. Macros may be nested up to 8 levels.
9. A maximum of 32 parameters may be passed in a macro or function call.
10. The maximum length of a line or macro definition is 8000 characters.

2.6.2 Siemens/Keil OMF66 Object Module Format Limits

1. Number of sections: 32767. Each compilation unit generates one code section (it is assumed that it contains at least one function) and a number of other sections (typically 1 to 8), depending on the memory types used in variable declarations.
2. Number of global (public) identifiers 32767. All file level data and function definitions not explicitly declared with the storage class `static` are public.
3. Number of external identifiers: 32767
4. Number of interrupt procedures: 128. This limit is imposed by the C167 architecture

3 Composition Of The Compiler Toolkit

The Keil C166 compiler toolkit consists of the following items:

Name	Revision	Description
C166	3.12j	‘C’ Compiler
A166	3.13	C166/7 Assembler
L166	3.13	Linker
LIB166	4.01	Librarian Tool
OH166	3.20	Object To Hex Converter
uVISION	1.32	Windows workbench & make utility

APPENDICES

4 Appendix A

4.1.1 Moving The USERSTACK On-Chip

The user stack is fixed in a section named ?C_CUSERSTACK, part of the NDATA class. However, it is quite common to place it in the IDATA class so that it can be on-chip. As the user stack is rarely very large, IDATA should be able to contain it.

A small modification is required to START167.A66 to move the USERSTACK into IDATA, as shown below:

START167.A66 Modified To Put User Stack In On-Chip RAM

The modifications required are:

Upper part of START166.A66 at line #479

```
?C_USERSTACK  SECTION DATA PUBLIC 'NDATA'
```

Should be modified to:

```
?C_USERSTACK  SECTION DATA PUBLIC 'IDATA'
;
```

... and further down the file...

```
; USTSZ: User Stack Size Definition
;  Defines the user stack space available for automatics.  This stack space is
;  accessed by R0.  The user stack space must be adjusted according the actual
;  requirements of the application.
USTSZ EQU     100H    ; set User Stack Size to 40H Bytes.
;
```

... and further down the file, after the EINIT instruction at line #700...

Make R0 be loaded with DPP3 for an on-chip USER STACK, rather than DPP2, as at present...

```
      ............EINIT
$IF (NOT TINY)
             MOV     R0,#DPP3:?C_USERSTKTOP   ; This was DPP2:?C_USERSTKTOP
$ENDIF
$IF TINY
      MOV     R0,#?C_USERSTKTOP
$ENDIF
```

If your have forgotten to reduce the size of the user stack to something small enough to fit in the on-chip RAM - the default 1000H bytes will cause L166 to issue a warning about the "IDATA class being out of group range".

Note: The USERSTACKDPP3 and NOFIXDPP C166 compiler controls must be used if the userstack is moved on-chip. This is to allow the compiler to correctly produce pointers to data that is on this stack. Failure to do this will result in undefined results.

4.1.1.1 The System Stack

With the user stack taking care of function parameters and local variables, the system stack is used for storing return addresses, the current PSW and CP plus any general purpose registers in the current register bank used for local register variables. This stack is always located on-chip and defaults to 256 words in length (80C166) at 0xfbff down to 0xfa00. The required stack size is set in the START167.A66 file. Values of 32, 64, 128 and 256 words can be selected via SYSCON. The CPU register “SP” has its top 5 bits hard-wired to ‘1’, the stack is always in the range 0xf800 to 0xfffe, i.e. on-chip.

4.1.1.2 Setting The System Stack Size

```
; STKSZ: Maximum System Stack Size selection  (SYSCON.13 .. SYSCON.14)
_STKSZ        EQU     0      ; System stack sizes
;                     ; 0 = 256 words (Reset Value)
;                     ; 1 = 128 words
;                     ; 2 =  64 words
;                     ; 3 =  32 words
```

The C166 is endowed with two special registers, STKOV and STKUN, which set the top and bottom limits of the stack. The default value of STKOV (Stack overflow) is 0xfa00 whilst STKUN (Stack underflow) defaults to 0xf00, in-line with the default 256 words.

The address of the stack is defined by loading the STKOV register is startup.a66

```
Setup Stack Overflow
_TOS   EQU     0FC00H                      ;top of system stack
_BOS   EQU     _TOS - (512 >> _STKSZ)    ;bottom of system stack

PUBLIC        ?C_SYSSTKBOT
?C_SYSSTKBOT  EQU     _BOS

              MOV     STKOV,#(_BOS+6*2)      ;INITIALIZE STACK OVFL REGISTER
```

L166 automatically reserves the appropriate on-chip memory and so no special actions are required by the user.

4.1.1.3 Setting Up The BUSCONx ADDRSELx Registers

The operation of BUSCON2,3,4 and ADDRSEL2,3,4 on the C165/7 is identical to the BUSCON1 and ADDRSEL1 on the C166. The big difference to the C166 is that there is a chip select pin for each ADDRSEL. Thus, if an address is accessed in a region covered by an ADDRSELx/BUSCONx pair, the corresponding chip select (CSx) pin goes low to enable the appropriate memory (or other) device.

It is essential that the ADDRSELx registers are initialised before the corresponding BUSCONx. This is to some extent common sense; the BUSCONx contains a BUSACT (bus active) bit which activates the bus characteristics over the preset ADDRSEL range. Unexpected results can occur if the BUSCONx is configured first for the following reasons:

(i) The C167/5 data book warns that no two ADDRSEL registers must describe an overlapping region. As all the ADDRSELs are set to zero when coming out of RESET, enabling two BUSCONs will cause an overlapping condition.

(ii) The BUSCONx region bus characteristics may differ from those of the background SYSCON. If the BUSCONx is enabled while the ADDRSELx is set to zero, the area currently executing could be changed.

Note1: The user must verify that the START167.A66 configures the ADDRSEL and BUSCON registers in the order given above.

Note2: The use must ensure that the BUSCONx and ADDRSELx for the currently executing region are not inadvertently changed.

The code to initialise the BUSCONx and ADDRSELx must be placed in the START167.A66, just after the BFLDH and BFLDL instructions that set up BUSCON0. It is not sensible to put the BUSCONx set up in C as any RAM areas described by a BUSCONx will not enabled and hence be zeroed or otherwise initialised by the C_STARTUP code in START167.A66.

```
 ?C_RESET     PROC TASK C_STARTUP INTNO RESET = 0
 ?C_STARTUP:

 $IF (WATCHDOG = 0)
                    DISWDT                 Disable watchdog timer
 $ENDIF

 BCON0L             SET     (_MTTC0 << 5) OR (_RWDC0 << 4) OR ((NOT _MCTC0) AND 0FH)
 BCON0L             SET    BCON0L AND (NOT (_RDYEN0 << 2))
 BCON0L             SET    BCON0L OR (_RWDC0 << 4) OR (_MTTC0 << 5)
 BCON0H             SET    (_ALECTL0 << 1) OR (_BUSACT0 << 2) OR (_RDYEN0 << 4)

                    BFLDL  BUSCON0,#3FH,#BCON0L
                    BFLDH  BUSCON0,#17H,#BCON0H

 ; **** Add ADDRSEL and BUSCON setups here! ****

                    MOV    ADDRSEL1,#421H
                    MOV    ADDRSEL2,#421H
             MOV    BUSCON1,#421H
                    MOV    BUSCON2,#421H
SYS_H        SET    (_STKSZ << 5) OR (_ROMS1 << 4) OR (_SGTEN << 3)
SYS_H        SET    SYS_H OR (_ROMEN << 2) OR (_BYTDIS << 1) OR _CLKEN
; Setup SYSCON Register
```

It is possible to create pointers to local data (i.e. automatics), although it is not really good practice. If the user stack has been moved into the IDATA on-chip RAM, you must use the USERSTACKDPP3 compilation control. This will force C166 to use DPP3 as the base for the pointer so that it can happily point to the user stack in IDATA rather than via DPP2 into the NDATA area.

```
#pragma USERSTACKDPP3   // Force compiler to use DPP3 when finding
                               // address of local array
void func1(char a, char b) {

char rx_buffer[0x20] ; // This array will be on user stack
        char *rx_ptr ;

        rx_ptr = &rx_buffer[a] ;  // Point to ath object in local array
```

5 Appendix B

5.1 Pointer Casting And Conversions

It is entirely possible to turn a simple C data type (int, long) into a pointer. However, the user must be aware of the number of bytes required to form the pointer in each case:

near pointers

One word carrying information on offset from DPP2 page number, 0-0x3FFF

far, huge, xhuge pointers

Two words carrying information on the page number in the upper word and offset from page base in the lower.

sdata pointers

One word carrying information on offset from page number 3 (DPP3), 0xC000-0xFFFF

5.1.1 Casting From Basic Types To Pointers

To make a near pointer, an `unsigned int` can be used:

Example

```
int near near_address = 0x4000 ;
int y = 0x8000 ;

y = *(unsigned char near * ) near_address ;
```

You must make very sure though with conversions to near pointers that the address is actually in the near data area!

To make a `far` pointers, the source data must be of type long as only this type has sufficient bytes to accommodate the pagenumber:offset information.

Example

```
unsigned long address = 0x38010 ;
unsigned char test ;

test = *(unsigned char far * ) address ;
```

It should be noted that this on-the-fly casting from long to pointer is very inefficient and it is always better to use a proper pointer type, as the following shows:

```
int var ;
unsigned long faddr = 0x30010 ;
int far *fptr = (int far *) 0x30010 ;
```

Using a proper pointer type:

```
var = *fptr ;
      MOV    R5,WORD fptr+2
      MOV    DPP0,R5
      MOV    R4,WORD fptr
      MOV    R4,[R4]
      MOV    WORD var,R4
```

Cast from long to pointer:

```
;    var =  *(int far*) faddr ;
      MOV    R4,WORD faddr
      MOV    R5,WORD faddr+2
      ADD    R4,R4
      ADDC   R5,R5
      ADD    R4,R4
      ADDC   R5,R5
      MOV    DPP0,R5
      SHR    R4,#2
      MOV    R8,[R4]
      MOV    DPP0,#12
```

Note: The conversion from long to a pointer is not like industry-standard PC compilers such as Microsoft C (MSC) for the 80x86. The calculation of the 166's page number is made during the cast at run-time, hence all the extra code produced above. In MSC, the long must have the segment number already in the right place. In the above example, rather than containing 0x30010, the long would have to have held 0x3000 0010. This could cause problems when converting MSC programs to the 166.

6 Appendix C

6.1 Pointers In The C166 Compiler

Pointers to data in the near data area unsurprisingly called termed "near pointers". They may point at any object in the near data area and as might be expected from the 16k pages, no single object may be over 16k. A near pointer in C166 is itself 16-bits, with the top two bits fixed at 02 to indicate DPP2.

6.2 The Most Common Pointers In C166

6.2.1 `far` Pointers

far pointers are less restricted in that the object being pointed at can be at any 18- or 24-bit address but the size of the object being pointed at must be limited to 16kb. As the value of DPP0 is calculated each time the pointer is used, far pointers are thus somewhat slower than near pointers. The 16kb limit only causes problems the user attempts to increment a pointer over this range, as might happen when using a pointer to access a large array. The reason for the limit is that when incrementing a far pointer, when the offset exceeds 0x3FFF, the DPP0 is not incremented and the offset simply wraps-around to zero again. Far pointers occupy 32-bits (two words)

6.2.2 huge Pointers

huge pointers (and objects) can be up to 64k in size as the overflow into the next page is *not* catered for. However, as C166 does not allow an overflow from a 16- to 24-bit offset, huge pointers will just wrap-around once they have been incremented more than 64kb from their start-point.

6.2.3 xhuge Pointers

xhuge pointers remove the 64kb limitation and allow objects of any size to be addressed without restriction. They are however relatively slow, unless you are using the C167/5.

6.2.4 `sdata` Pointers

The final pointer type, **sdata**, is C166 family-specific. This is a pointer which is always points into the system area, indicated by DPP3, between 0xC000 and 0xFFFF. sdata pointers are 16-bits in size and are best used for pointing at internal RAM objects or IO mapped into the 0xC000 region.

6.3 Summary Of Pointer Declarations

Declare a near pointer:

int near *near_ptr ;

Declare a far pointer:

```
int far *far_ptr ;
```

Declare a sdata pointer:

```
int sdata *s_ptr ;
```

6.4 Special Note On #pragma MOD167 For C167 CPUs

The #pragma MOD167 is used for huge basic types and pointer accesses by using the EXTS seg,off instruction.

MOD167 Huge Pointer Access

```
;       *hptr = 0x55 ;
                                            ; SOURCE LINE # 19
      MOV     R6,#85
      MOV     R5,WORD hptr+2
      MOV     R4,WORD hptr
      EXTS    R5,#1
      MOV     [R4],R6
```

R5 holds the 64kb segment number and R6 holds the offset into the segment. In effect, this a full 32-bit access being equivalent to MOV [R5:R4],R6.

7 Appendix D

7.1 BSI Definition Of Year 2000 Compliance – Keil C166 Tools

The following Keil tools of equal or later version number to those listed below are Year 2000 compliant according to the DISC PD2000-1 definition (follows):

C51 Compiler for 8051	V5.50	A51 Assembler	V5.50	BL51 Linker	V3.70
uVision-51 Windows	V1.31	DScope51 for Windows	V1.5	OH51	V1.2
C166 Compiler	V3.11	A166 Assembler	V3.12	L166 Linker	V3.11
uVision-166 Windows	V1.31	DScope166 for Windows	V1.02	OH166	V2.0
C251 Compiler	V2.12a	A251 Assembler	V1.4	L251 Linker	V1.29
uVision-251 Windows	V1.31	DScope251 for Windows	V1.3w	OH251	V1.4

Notes:

(i) Earlier versions of compilers will continue to function but date-related operation may not return the correct values.

(ii) Older projects that are in the support and maintenance phase using earlier versions of Keil compilers may suffer from Year 2000 problems if they rely on the __DATE__ intrinsic macro. Other problems may arise and you should make a full audit of older projects to see what the implications of this are.

(iii) Neither Keil nor Hitex can be held responsible for the consequences arising from the continued use of tool versions of earlier release numbers than those listed above.

A DEFINITION OF YEAR 2000 CONFORMITY REQUIREMENTS

Introduction

This document addresses what is commonly known as Year 2000 Conformity (also sometimes known as century or millennium compliance). It provides a definition of this expression and requirements that must be satisfied in equipment and products, which use dates and times.

It has been prepared by British Standards Institution committee BDD/1/-/3 in response to demand from UK industry, commerce and the public sector. It is the result of work from the following bodies whose contributions are gratefully acknowledged: BT, CapGemini, CCTA, Coopers & Lybrand, Halberstam Elias, ICL, National Health Service, National Westminster Bank.

BSI-DISC would also like to thank the following organisations for their support and encouragement in the development of this definition: taskforce 2000, Barclays Bank, British Airways, Cambridgeshire County Council, Computer Software Services Association, Department of Health, Ernst & Young, Federation of Small Businesses, IBM, ICI, National Power, Paymaster Agency, Prudential Assurance, Reuters, Tesco Stores.

While every care has been taken in developing this document, the contributing organisations accept no liability for any loss or damage caused, arising directly or indirectly, in connection with reliance on its contents except to the extent that such liability may not be excluded at law. Independent legal advice should be sought by any person or organisation intending to enter into a contractual commitment relating to Year 2000 conformity requirements.

THE DEFINITION

Year 2000 conformity shall mean that neither performance nor functionality is affected by dates prior to, during and after the year 2000.

In particular:
Rule 1. No value for current date will cause any interruption in operation.

Rule 2. Date-based functionality must behave consistently for dates prior to, during and after year 2000.

Rule 3. In all interfaces and data storage, the century in any date must be specified either explicitly or by unambiguous algorithms or inferencing rules.

Rule 4. Year 2000 must be recognised as a leap year.

AMPLIFICATION OF THE DEFINITION AND RULES

General Explanation

Problems can arise from some means of representing dates in computer equipment and products and from date-logic embedded in purchased goods or services, as the year 2000 approaches and during and after that year. As a result, equipment or products, including embedded control logic, may fail completely, malfunction or cause data to be corrupted.

To avoid such problems, organisations must check, and modify if necessary, internally produced equipment and products and similarly check externally supplied equipment and products with their suppliers. The purpose of this document is to allow such checks to be made on a basis of common understanding.

Where checks are made with external suppliers, care should betaken to distinguish between claims of conformity and the ability to demonstrate conformity.

Rule 1

1.1 This rule is sometimes known as general integrity.

1.2 If this requirement is satisfied, roll-over between all significant time demarcations (e.g. days, months, years, centuries) will be performed correctly.

1.3 Current date means today's date as known to the equipment or product.

Rule 2

2.1 This rule is sometimes known as date integrity.
2.2 This rule means that all equipment and products must calculate, manipulate and represent dates correctly for the purposes for which they were intended.
2.3 The meaning of functionality includes both processes and the results of those processes.

2.4 If desired, a reference point for date values and calculations may be added by organisations; e.g. as defined by the Gregorian calendar.

2.5 No equipment or product shall use particular date values for special meanings; e.g. "99" to signify "no end value" or "end of file" or "00" to mean "not applicable" or "beginning of file".

Rule 3

3.1 This rule is sometimes known as explicit/implicit century.
3.2 It covers two general approaches:

(a) explicit representation of the year in dates: e.g. by using four digits or by including a century indicator. In this case, a reference may be inserted (e.g. 4-digit years as allowed by ISO standard 8601:1988) and it may be necessary to allow for exceptions where domain-specific standards (e.g. standards relating to Electronic Data Interchange, Automatic Teller Machines or Bankers Automated Clearing Services) should have precedence.

(b) the use of inferencing rules: e.g. two-digit years with a value greater than 50 imply 19xx, those with a value equal to or less than 50 imply 20xx. Rules for century inferencing as a whole must apply to all contexts in which the date is used, although different inferencing rules may apply to different datesets.

General Notes

For Rules 1 and 2 in particular, organisations may wish to specify allowable ranges for values of current date and dates to be manipulated. The ranges may relate to one or more of the feasible life-span of equipment or products or the span of dates required to be represented by the organisation's business processes. Tests for specifically critical dates may also be added (e.g. for leap years, end of year, etc). Organisations may wish to append additional material in support of local requirements.

Where the term century is used, clear distinction should be made between the "value" denoting the century (e.g. 20th) and its representation in dates (e.g. 19xx); similarly, 21st and 20xx.

ISBN 0 580 29746 2

DISC is a part of the British Standards Institution

BSI, 389 Chiswick High Road, London W4 4AL

Tel: 0181 996 9000

8 Appendix E

8.1 Results Log Of PARANOIA Test For KEIL C166 v3.12j

The following test was captured from Windows Hyperterm terminal via the clipboard. It was emitted from the C167 serial port at 9600 baud.

```
Lest this program stop prematurely, i.e. before displaying

    `END OF TEST',

try to persuade the computer NOT to terminate execution when an
error like Over/Underflow or Division by Zero occurs, but rather
to persevere with a surrogate value after, perhaps, displaying some
warning.  If persuasion avails naught, don't despair but run this
program anyway to see how many milestones it passes, and then
amend it to make further progress.

Answer questions with Y, y, N or n (unless otherwise indicated).


To continue, press RETURN
Diagnosis resumes after milestone Number 0          Page: 1

Users are invited to help debug and augment this program so it will
cope with unanticipated and newly uncovered arithmetic pathologies.

Please send suggestions and interesting results to
        Richard Karpinski
        Computer Center U-76
        University of California
        San Francisco, CA 94143-0704, USA

In doing so, please include the following information:
        Precision:      single;
        Version:        10 February 1989;
        Computer:

        Compiler:

        Optimization level:

        Other relevant compiler options:

To continue, press RETURN
Diagnosis resumes after milestone Number 1          Page: 2

Running this program should reveal these characteristics:
     Radix = 1, 2, 4, 8, 10, 16, 100, 256 ...
     Precision = number of significant digits carried.
     U2 = Radix/Radix^Precision = One Ulp
        (OneUlpnit in the Last Place) of 1.000xxx .
     U1 = 1/Radix^Precision = One Ulp of numbers a little less than 1.0 .
     Adequacy of guard digits for Mult., Div. and Subt.
     Whether arithmetic is chopped, correctly rounded, or something else
        for Mult., Div., Add/Subt. and Sqrt.
     Whether a Sticky Bit used correctly for rounding.
     UnderflowThreshold = an underflow threshold.
     E0 and PseudoZero tell whether underflow is abrupt, gradual, or fuzzy.
     V = an overflow threshold, roughly.
     V0  tells, roughly, whether  Infinity  is represented.
     Comparisions are checked for consistency with subtraction
        and for contamination with pseudo-zeros.
     Sqrt is tested.  Y^X is not tested.
     Extra-precise subexpressions are revealed but NOT YET tested.
     Decimal-Binary conversion is NOT YET tested for accuracy.

To continue, press RETURNLest this program stop prematurely, i.e. before display
```

```
ing

    `END OF TEST',

try to persuade the computer NOT to terminate execution when an
error like Over/Underflow or Division by Zero occurs, but rather
to persevere with a surrogate value after, perhaps, displaying some
warning.  If persuasion avails naught, don't despair but run this
program anyway to see how many milestones it passes, and then
amend it to make further progress.

Answer questions with Y, y, N or n (unless otherwise indicated).


To continue, press RETURN
Diagnosis resumes after milestone Number 0          Page: 1

Users are invited to help debug and augment this program so it will
cope with unanticipated and newly uncovered arithmetic pathologies.

Please send suggestions and interesting results to
        Richard Karpinski
        Computer Center U-76
        University of California
        San Francisco, CA 94143-0704, USA

In doing so, please include the following information:
        Precision:      single;
        Version:        10 February 1989;
        Computer:

        Compiler: Keil C166 v3.12j

        Optimization level: OT(6)

        Other relevant compiler options: HLARGE, MOD167

To continue, press RETURN
Diagnosis resumes after milestone Number 1          Page: 2

Running this program should reveal these characteristics:
     Radix = 1, 2, 4, 8, 10, 16, 100, 256 ...
     Precision = number of significant digits carried.
     U2 = Radix/Radix^Precision = One Ulp
        (OneUlpnit in the Last Place) of 1.000xxx .
     U1 = 1/Radix^Precision = One Ulp of numbers a little less than 1.0 .
     Adequacy of guard digits for Mult., Div. and Subt.
     Whether arithmetic is chopped, correctly rounded, or something else
        for Mult., Div., Add/Subt. and Sqrt.
     Whether a Sticky Bit used correctly for rounding.
     UnderflowThreshold = an underflow threshold.
     E0 and PseudoZero tell whether underflow is abrupt, gradual, or fuzzy.
     V = an overflow threshold, roughly.
     V0  tells, roughly, whether  Infinity  is represented.
     Comparisions are checked for consistency with subtraction
        and for contamination with pseudo-zeros.
     Sqrt is tested.  Y^X is not tested.
     Extra-precise subexpressions are revealed but NOT YET tested.
     Decimal-Binary conversion is NOT YET tested for accuracy.

To continue, press RETURN
Diagnosis resumes after milestone Number 2          Page: 3

The program attempts to discriminate among
   FLAWs, like lack of a sticky bit,
   Serious DEFECTs, like lack of a guard digit, and
   FAILUREs, like 2+2 == 5 .
Failures may confound subsequent diagnoses.

The diagnostic capabilities of this program go beyond an earlier
program called `MACHAR', which can be found at the end of the
book  `Software Manual for the Elementary Functions' (1980) by
W. J. Cody and W. Waite. Although both programs try to discover
the Radix, Precision and range (over/underflow thresholds)
of the arithmetic, this program tries to cope with a wider variety
of pathologies, and to say how well the arithmetic is implemented.
```

```
The program is based upon a conventional radix representation for
floating-point numbers, but also allows logarithmic encoding
as used by certain early WANG machines.

BASIC version of this program (C) 1983 by Prof. W. M. Kahan;
see source comments for more history.

To continue, press RETURN
Diagnosis resumes after milestone Number 3          Page: 4

Program is now RUNNING tests on small integers:
-1, 0, 1/2, 1, 2, 3, 4, 5, 9, 27, 32 & 240 are O.K.

Searching for Radix and Precision.
Radix = 2.000000 .
Closest relative separation found is U1 = 5.9604640e-08 .

Recalculating radix and precision
 confirms closest relative separation U1 .
Radix confirmed.
FAILURE:  (1-U1)-1/2 < 1/2 is FALSE, prog. fails?.
FAILURE:  Comparison is fuzzy,X=1 but X-1/2-1/2 != 0.
The number of significant digits of the Radix is 24.000000 .

To continue, press RETURN
Diagnosis resumes after milestone Number 30          Page: 5

Subtraction appears to be normalized, as it should be.
Checking for guard digit in *, /, and -.
     *, /, and - appear to have guard digits, as they should.

To continue, press RETURN
Diagnosis resumes after milestone Number 40          Page: 6

Checking rounding on multiply, divide and add/subtract.
Multiplication appears to round correctly.
/ is neither chopped nor correctly rounded.
Addition/Subtraction neither rounds nor chops.
Sticky bit used incorrectly or not at all.
FLAW:  lack(s) of guard digits or failure(s) to correctly round or chop
(noted above) count as one flaw in the final tally below.

Does Multiplication commute?  Testing on 20 random pairs.
     No failures found in 20 integer pairs.

Running test of square root(x).
Testing if sqrt(X * X) == X for 20 Integers X.
Test for sqrt monotonicity.
sqrt has passed a test for Monotonicity.
Testing whether sqrt is rounded or chopped.
Square root is neither chopped nor correctly rounded.
Observed errors run from 0.0000000e+00 to 5.0000000e-01 ulps.

To continue, press RETURN
Diagnosis resumes after milestone Number 90          Page: 7

Testing powers Z^i for small Integers Z and i.
WARNING:  computing
        (0.00000000000000000e+00) ^ (0.00000000000000000e+00)
        yielded ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ;
        which compared unequal to correct 1.00000000000000000e+00 ;
                they differ by ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ .
Similar discrepancies have occurred 5 times.
DEFECT:  computing
        (5.00000000000000000e-01) ^ (2.60000000000000000e+01)
        yielded 1.49011400000000000e-08;
        which compared unequal to correct 1.49011600000000000e-08 ;
                they differ by -2.04281000000000000e-14 .
Errors like this may invalidate financial calculations
        involving interest rates.
Similar discrepancies have occurred 14 times.

To continue, press RETURN
Diagnosis resumes after milestone Number 100          Page: 8
```

```
Seeking Underflow thresholds UfThold and E0.
FAILURE:  multiplication gets too many last digits wrong.

To continue, press RETURN
Diagnosis resumes after milestone Number 110          Page: 9

Smallest strictly positive number found is E0 = 1.17549e-38 .
Since comparison denies Z = 0, evaluating (Z + Z) / Z should be safe.
What the machine gets for (Z + Z) / Z is  2.00000000000000000e+00 .
This is O.K., provided Over/Underflow has NOT just been signaled.

The Underflow threshold is 1.17549400000000000e-38,  below which
calculation may suffer larger Relative error than merely roundoff.
Since underflow occurs below the threshold
UfThold = (2.00000000000000000e+00) ^ (-1.26000000000000000e+02)
only underflow should afflict the expression
        (2.00000000000000000e+00) ^ (-2.52000000000000000e+02);
actually calculating yields: 0.00000000000000000e+00 .
This computed value is O.K.

Testing X^((X + 1) / (X - 1)) vs. exp(2) = 7.38905700000000000e+00 as X -> 1.
DEFECT:  Calculated 1.26641700000000000e-14 for
        (1 + (-5.96046400000000000e-08) ^ (-3.35544300000000000e+07);
        differs from correct value by -7.38905700000000000e+00 .
        This much error may spoil financial
        calculations involving tiny interest rates.
Testing powers Z^Q at four nearly extreme values.
 ... no discrepancies found.


To continue, press RETURN
Diagnosis resumes after milestone Number 190          Page: 10

DEFECT:  Badly unbalanced range; UfThold * V = 0.00000000000000000e+00
        is too far from 1.

SERIOUS DEFECT:    X / X differs from 1 when X = 0.00000000000000000e+00
  instead, X / X - 1/2 - 1/2 =  .

What message and/or values does Division by Zero produce?
This can interupt your program.  You can skip this part if you wish.
Do you wish to compute 1 / 0?     Trying to compute 1 / 0 produces ... 0 0.00000
00e+00 .

Do you wish to compute 0 / 0?
    Trying to compute 0 / 0 produces ... 0 0.0000000e+00 .

To continue, press RETURN
Diagnosis resumes after milestone Number 220          Page: 11


The number of  FAILUREs  encountered =       3.
The number of  SERIOUS DEFECTs  discovered = 1.
The number of  DEFECTs  discovered =         3.
The number of  FLAWs  discovered =           1.

The arithmetic diagnosed has unacceptable Serious Defects.
Potentially fatal FAILURE may have spoiled this program's subsequent diagnoses.
END OF TEST.
```

8.2 Results Log Of PARANOIA Test For Microsoft C v11.00

Configured for 386 code generation using Floating point Emulator

The following test was captured from a DOS window in Windows 95 via the clipboard.

```
Lest this program stop prematurely, i.e. before displaying

    `END OF TEST',

try to persuade the computer NOT to terminate execution when an
error like Over/Underflow or Division by Zero occurs, but rather
to persevere with a surrogate value after, perhaps, displaying some
warning.  If persuasion avails naught, don't despair but run this
program anyway to see how many milestones it passes, and then
amend it to make further progress.

Answer questions with Y, y, N or n (unless otherwise indicated).


To continue, press RETURN

Diagnosis resumes after milestone Number 0          Page: 1

Users are invited to help debug and augment this program so it will
cope with unanticipated and newly uncovered arithmetic pathologies.

Please send suggestions and interesting results to
        Richard Karpinski
        Computer Center U-76
        University of California
        San Francisco, CA 94143-0704, USA

In doing so, please include the following information:
        Precision:      single;
        Version:        10 February 1989;
        Computer:

        Compiler: MSC v8.00

        Optimization level:MAX

        Other relevant compiler options: -G3 for 386 code with no FPU

To continue, press RETURN

Diagnosis resumes after milestone Number 1          Page: 2

Running this program should reveal these characteristics:
     Radix = 1, 2, 4, 8, 10, 16, 100, 256 ...
     Precision = number of significant digits carried.
     U2 = Radix/Radix^Precision = One Ulp
        (OneUlpnit in the Last Place) of 1.000xxx .
     U1 = 1/Radix^Precision = One Ulp of numbers a little less than 1.0 .
     Adequacy of guard digits for Mult., Div. and Subt.
     Whether arithmetic is chopped, correctly rounded, or something else
        for Mult., Div., Add/Subt. and Sqrt.
     Whether a Sticky Bit used correctly for rounding.
     UnderflowThreshold = an underflow threshold.
     E0 and PseudoZero tell whether underflow is abrupt, gradual, or fuzzy.
     V = an overflow threshold, roughly.
     V0  tells, roughly, whether  Infinity  is represented.
     Comparisions are checked for consistency with subtraction
        and for contamination with pseudo-zeros.
     Sqrt is tested.  Y^X is not tested.
     Extra-precise subexpressions are revealed but NOT YET tested.
     Decimal-Binary conversion is NOT YET tested for accuracy.

To continue, press RETURN


Diagnosis resumes after milestone Number 2          Page: 3
```

```
The program attempts to discriminate among
   FLAWs, like lack of a sticky bit,
   Serious DEFECTs, like lack of a guard digit, and
   FAILUREs, like 2+2 == 5 .
Failures may confound subsequent diagnoses.

The diagnostic capabilities of this program go beyond an earlier
program called `MACHAR', which can be found at the end of the
book  `Software Manual for the Elementary Functions' (1980) by
W. J. Cody and W. Waite. Although both programs try to discover
the Radix, Precision and range (over/underflow thresholds)
of the arithmetic, this program tries to cope with a wider variety
of pathologies, and to say how well the arithmetic is implemented.

The program is based upon a conventional radix representation for
floating-point numbers, but also allows logarithmic encoding
as used by certain early WANG machines.

BASIC version of this program (C) 1983 by Prof. W. M. Kahan;
see source comments for more history.

To continue, press RETURN

Diagnosis resumes after milestone Number 3          Page: 4

Program is now RUNNING tests on small integers:
-1, 0, 1/2, 1, 2, 3, 4, 5, 9, 27, 32 & 240 are O.K.

Searching for Radix and Precision.
Radix = 1073741824.000000 .
Closest relative separation found is U1 = 9.3132257e-010 .

Recalculating radix and precision
 gets better closest relative separation U1 = 9.9341078e-009 .
MYSTERY: recalculated Radix = 1.0066330e+008 .
DEFECT:  Radix is too big: roundoff problems.
FLAW:  Radix is not as good as 2 or 10.
FAILURE:  (1-U1)-1/2 < 1/2 is FALSE, prog. fails?.
The number of significant digits of the Radix is 1.000000 .
SERIOUS DEFECT:  Precision worse than 5 decimal figures  .
Some subexpressions appear to be calculated extra
precisely with about 0.993608 extra B-digits, i.e.
roughly 7.95172 extra significant decimals.
That feature is not tested further by this program.

To continue, press RETURN

FAILURE:  Incomplete carry-propagation in Addition.
Add/Subtract appears to be chopped.
Addition/Subtraction neither rounds nor chops.
Sticky bit used incorrectly or not at all.
FLAW:  lack(s) of guard digits or failure(s) to correctly round or chop
(noted above) count as one flaw in the final tally below.

Does Multiplication commute?  Testing on 20 random pairs.
DEFECT:  X * Y == Y * X trial fails.

Running test of square root(x).
SERIOUS DEFECT:
sqrt( 1.01330991615836160e+016) - 1.00663296000000000e+008  = -1.000000000000000
00e+000
        instead of correct value 0 .
SERIOUS DEFECT:
sqrt( 9.86864969599650250e-017) - 9.93410775862457740e-009  = -9.868649843050824
90e-017
        instead of correct value 0 .
SERIOUS DEFECT:
sqrt( 9.86864969599650250e-017) - 9.93410775862457740e-009  = -9.868649843050824
90e-017
        instead of correct value 0 .

To continue, press RETURN

sqrt( 1.01330991615836160e+016) - 1.00663296000000000e+008  = -1.000000000000000
00e+000
```

```
        instead of correct value 0 .
SERIOUS DEFECT:
sqrt( 9.86864969599650250e-017) - 9.93410775862457740e-009  = -9.868649843050824
90e-017
        instead of correct value 0 .
SERIOUS DEFECT:
sqrt( 9.86864969599650250e-017) - 9.93410775862457740e-009  = -9.868649843050824
90e-017
        instead of correct value 0 .

To continue, press RETURN

Diagnosis resumes after milestone Number 70          Page: 7

Testing if sqrt(X * X) == X for 20 Integers X.
DEFECT:
sqrt( 1.01330991615836160e+016) - 1.00663296000000000e+008  = -1.000000029802322
40e+000
        instead of correct value 0 .
Test for sqrt monotonicity.
sqrt has passed a test for Monotonicity.
Testing whether sqrt is rounded or chopped.

(PROGRAM HANGS UP AT THIS POINT)


(PROGRAM MODIFIED TO REMOVE SQRT TEST – OUTPUT LOGGING RESUMED)

Diagnosis resumes after milestone Number 70          Page: 7

Testing if sqrt(X * X) == X for 20 Integers X.
DEFECT:
sqrt( 1.01330991615836160e+016) - 1.00663296000000000e+008  = -1.000000029802322
40e+000
        instead of correct value 0 .
Test for sqrt monotonicity.
sqrt has passed a test for Monotonicity.

To continue, press RETURN



Diagnosis resumes after milestone Number 90          Page: 8

Testing powers Z^i for small Integers Z and i.
DEFECT:  computing
        (1.00000001490116120e-001) ^ (7.00000000000000000e+000)
        yielded 1.00000008274037100e-007;
        which compared unequal to correct 1.00000015379464460e-007 ;
                they differ by -7.10542735760100190e-015 .
Errors like this may invalidate financial calculations
        involving interest rates.
Similar discrepancies have occurred 10 times.

To continue, press RETURN

calculation may suffer larger Relative error than merely roundoff.
DEFECT:  Range is too narrow; U1^5 Underflows.
Since underflow occurs below the threshold
UfThold = (1.00663296000000000e+008) ^ (-3.96250009536743160e+000)
only underflow should afflict the expression
        (1.00663296000000000e+008) ^ (-7.92500019073486330e+000);
actually calculating yields: 0.00000000000000000e+000 .
This computed value is O.K.

Testing X^((X + 1) / (X - 1)) vs. exp(2) = 7.38905572891235350e+000 as X -> 1.
DEFECT:  Calculated 2.67379508591339170e+002 for
        (1 + (2.55000000000000000e+002) ^ (1.00784313678741460e+000);
        differs from correct value by 2.59990447998046870e+002 .
        This much error may spoil financial
        calculations involving tiny interest rates.
Testing powers Z^Q at four nearly extreme values.
DEFECT:  computing
        (1.00000000000000000e+001) ^ (3.20000000000000000e+001)
        yielded 1.00000003318135350e+032;
        which compared unequal to correct 1.02679679275673470e+032 ;
```

```
                they differ by -2.67967595753811630e+030 .
Similar discrepancies have occurred 4 times.


To continue, press RETURN


DEFECT:  Calculated 2.67379508591339170e+002 for
        (1 + (2.55000000000000000e+002) ^ (1.00784313678741460e+000);
        differs from correct value by 2.59990447998046870e+002 .
        This much error may spoil financial
        calculations involving tiny interest rates.
Testing powers Z^Q at four nearly extreme values.
DEFECT:  computing
        (1.00000000000000000e+001) ^ (3.20000000000000000e+001)
        yielded 1.00000003318135350e+032;
        which compared unequal to correct 1.02679679275673470e+032 ;
                they differ by -2.67967595753811630e+030 .
Similar discrepancies have occurred 4 times.


To continue, press RETURN

Diagnosis resumes after milestone Number 190          Page: 10

DEFECT:  Badly unbalanced range; UfThold * V = 0.00000000000000000e+000
        is too far from 1.


What message and/or values does Division by Zero produce?
This can interupt your program.  You can skip this part if you wish.
Do you wish to compute 1 / 0?

(REPLY YES)

This much error may spoil financial
        calculations involving tiny interest rates.
Testing powers Z^Q at four nearly extreme values.
DEFECT:  computing
        (1.00000000000000000e+001) ^ (3.20000000000000000e+001)
        yielded 1.00000003318135350e+032;
        which compared unequal to correct 1.02679679275673470e+032 ;
                they differ by -2.67967595753811630e+030 .
Similar discrepancies have occurred 4 times.


To continue, press RETURN

Diagnosis resumes after milestone Number 190          Page: 10

DEFECT:  Badly unbalanced range; UfThold * V = 0.00000000000000000e+000
        is too far from 1.


What message and/or values does Division by Zero produce?
This can interupt your program.  You can skip this part if you wish.
Do you wish to compute 1 / 0? y
    Trying to compute 1 / 0 produces ...  1.#INF000e+000 .

Do you wish to compute 0 / 0?

(REPLY YES)

(1.00000000000000000e+001) ^ (3.20000000000000000e+001)
        yielded 1.00000003318135350e+032;
        which compared unequal to correct 1.02679679275673470e+032 ;
                they differ by -2.67967595753811630e+030 .
Similar discrepancies have occurred 4 times.


To continue, press RETURN

Diagnosis resumes after milestone Number 190          Page: 10

DEFECT:  Badly unbalanced range; UfThold * V = 0.00000000000000000e+000
        is too far from 1.
```

```
What message and/or values does Division by Zero produce?
This can interupt your program.  You can skip this part if you wish.
Do you wish to compute 1 / 0? y
    Trying to compute 1 / 0 produces ...  1.#INF000e+000 .

Do you wish to compute 0 / 0? y

    Trying to compute 0 / 0 produces ...  -1.#IND000e+000 .

To continue, press RETURN



What message and/or values does Division by Zero produce?
This can interupt your program.  You can skip this part if you wish.
Do you wish to compute 1 / 0? y
    Trying to compute 1 / 0 produces ...  1.#INF000e+000 .

Do you wish to compute 0 / 0? y

    Trying to compute 0 / 0 produces ...  -1.#IND000e+000 .

To continue, press RETURN

Diagnosis resumes after milestone Number 220          Page: 11


The number of  FAILUREs  encountered =       6.
The number of  SERIOUS DEFECTs  discovered = 5.
The number of  DEFECTs  discovered =         10.
The number of  FLAWs  discovered =           2.

The arithmetic diagnosed has unacceptable Serious Defects.
Potentially fatal FAILURE may have spoiled this program's subsequent diagnoses.
END OF TEST.
```

9 Appendix F

9.1 Plum-Hall test summary files.

9.1.1 Invocation Batch File

```
@echo off
rem  compiler  - compile source file  %1  with include-dirs  %2 %3 ...

rem ###logic     you must configure all this script for your compiler

rem              remove old object file, in case something abends
if exist %1%OBJ% del %1%OBJ%
rem              start a new "clg" (compiler log) file
echo --- compile %1 --- >%1.clg
rem              choose the compile recipe according to number of args
if "%2" == "" goto notwo
if "%3" == "" goto nothree
if "%4" == "" goto nofour

if exist phstatus del phstatus

rem set INCLUDE= ... set external  INCLUDE  if compiler uses one ...
rem ###format     you need to hand-configure the compile commands ...
set %PHIN%=%PHIP%;%2;%3;%4;%PHDST%;%PHSRC%\conform
%PHCC% %2%1.c OJ(%1%OBJ%) %CFLAGS% >>%1.clg
goto done
:nofour
set %PHIN%=%PHIP%;%2;%3;%PHDST%;%PHSRC%\conform
%PHCC% %2%1.c OJ(%1%OBJ%) %CFLAGS% >>%1.clg
goto done
:nothree
set %PHIN%=%PHIP%;%2;%PHDST%;%PHSRC%\conform
%PHCC% %2%1.c OJ(%1%OBJ%) %CFLAGS% >>%1.clg
goto done
:notwo
rem              using only one arg is allowed only if PHSRC == PHDST
if %PHSRC% == %PHDST% goto onetree
echo usage:  %0 pgm-name  source-dir [ include-dir ]
echo The  source-dir  is required if you are compiling remote sources.
goto done
:onetree
set %PHIN%=%PHIP%;%PHDST%;%PHSRC%\conform
%PHCC% %1.c OJ(%1%OBJ%) %CFLAGS% >>%1.clg
:done

if errorlevel 1 echo 1 >phstatus

rem Any tests for success-or-failure of the compile command need
rem to test %PHSTATUS%  .

rem You could execute the "diagnost" command here to display your
rem diagnostic messages.

rem diagnost %1.clg

exit 0
```

9.1.2 Summary File 1

```
EXPECTED  ACTUAL  ERRORS SKIPPED  FILE NAME     5.00  1994-01-03
      16      16       0       0   conform/environ.out
      24      24       0       4   conform/interp91.out
    1224    1223       1       0 **conform/lang.out
     123    7208       0      16 **conform/libfrst.out
     821     821       0       0   conform/prec1.out
    1180    1180       0       0   conform/prec2.out
       1       1       0       0   conform/capacity/capacity.out
      11      11       0       0   conform/errauto/i91.out
     151     151       0       0   conform/errauto/j91.out
      32      31       1       0 **conform/errauto/m61.out
       0       0       0       0   conform/errauto/m62.out
      75      75       0       0   conform/errauto/m63.out
       5       4       1       0 **conform/errauto/m64.out
      77      76       1       0 **conform/errauto/m65.out
      24      24       0       0   conform/errauto/m66.out
      17      17       0       0   conform/errauto/m67.out
      39      39       0       0   conform/errauto/m68.out
     288     288       0       0   conform/exprtest/andif.out
     576     576       0       0   conform/exprtest/assign.out
     162     162       0       0   conform/exprtest/band.out
     324     324       0       0   conform/exprtest/bandeq.out
     288     288       0       0   conform/exprtest/cast.out
     162     162       0       0   conform/exprtest/compl.out
     288     288       0       0   conform/exprtest/div.out
     120     120       0       0   conform/exprtest/diveq1.out
     156     156       0       0   conform/exprtest/diveq2.out
     156     156       0       0   conform/exprtest/diveq3.out
     144     144       0       0   conform/exprtest/diveq4.out
     288     288       0       0   conform/exprtest/eq.out
     288     288       0       0   conform/exprtest/ge.out
     288     288       0       0   conform/exprtest/gt.out
      13      13       0       0   conform/exprtest/int1.out
       9       9       0       0   conform/exprtest/int2.out
       9       9       0       0   conform/exprtest/int3.out
      12      12       0       0   conform/exprtest/int4.out
      13      13       0       0   conform/exprtest/int5.out
      12      12       0       0   conform/exprtest/int6.out
      13      13       0       0   conform/exprtest/int7.out
      15      15       0       0   conform/exprtest/int8.out
      13      13       0       0   conform/exprtest/int9.out
      10      10       0       0   conform/exprtest/int10.out
      13      13       0       0   conform/exprtest/int11.out
      14      14       0       0   conform/exprtest/int12.out
      11      11       0       0   conform/exprtest/int13.out
     288     288       0       0   conform/exprtest/le.out
     162     162       0       0   conform/exprtest/lsh.out
     324     324       0       0   conform/exprtest/lsheq.out
     288     288       0       0   conform/exprtest/lt.out
     122     122       0       0   conform/exprtest/mineq1.out
     154     154       0       0   conform/exprtest/mineq2.out
     152     152       0       0   conform/exprtest/mineq3.out
     136     136       0       0   conform/exprtest/mineq4.out
     288     288       0       0   conform/exprtest/minus.out
      12      12       0       0   conform/exprtest/mix1.out
       9       9       0       0   conform/exprtest/mix2.out
       9       9       0       0   conform/exprtest/mix3.out
      12      12       0       0   conform/exprtest/mix4.out
       9       9       0       0   conform/exprtest/mix5.out
      14      14       0       0   conform/exprtest/mix6.out
      11      11       0       0   conform/exprtest/mix7.out
      17      17       0       0   conform/exprtest/mix8.out
      11      11       0       0   conform/exprtest/mix9.out
      12      12       0       0   conform/exprtest/mix10.out
      15      15       0       0   conform/exprtest/mix11.out
      16      16       0       0   conform/exprtest/mix12.out
      14      14       0       0   conform/exprtest/mix13.out
     288     288       0       0   conform/exprtest/ne.out
     288     288       0       0   conform/exprtest/not.out
     162     162       0       0   conform/exprtest/or.out
     288     288       0       0   conform/exprtest/orelse.out
     324     324       0       0   conform/exprtest/oreq.out
     122     122       0       0   conform/exprtest/plueq1.out
     156     156       0       0   conform/exprtest/plueq2.out
     154     154       0       0   conform/exprtest/plueq3.out
     144     144       0       0   conform/exprtest/plueq4.out
     288     288       0       0   conform/exprtest/plus.out
     576     576       0       0   conform/exprtest/postdec.out
     576     576       0       0   conform/exprtest/preinc.out
     576     576       0       0   conform/exprtest/quest.out
      12      12       0       0   conform/exprtest/real1.out
      11      11       0       0   conform/exprtest/real2.out
      13      13       0       0   conform/exprtest/real3.out
      11      11       0       0   conform/exprtest/real4.out
      10      10       0       0   conform/exprtest/real5.out
      12      12       0       0   conform/exprtest/real6.out
      15      15       0       0   conform/exprtest/real7.out
      11      11       0       0   conform/exprtest/real8.out
       8       8       0       0   conform/exprtest/real9.out
      15      15       0       0   conform/exprtest/real10.out
      10      10       0       0   conform/exprtest/real11.out
      16      16       0       0   conform/exprtest/real12.out
      15      15       0       0   conform/exprtest/real13.out
     162     162       0       0   conform/exprtest/remain.out
     324     324       0       0   conform/exprtest/remeq.out
     162     162       0       0   conform/exprtest/rsh.out
     324     324       0       0   conform/exprtest/rsheq.out
     120     120       0       0   conform/exprtest/timeq1.out
     154     154       0       0   conform/exprtest/timeq2.out
     156     156       0       0   conform/exprtest/timeq3.out
     146     146       0       0   conform/exprtest/timeq4.out
     288     288       0       0   conform/exprtest/timesop.out
     288     288       0       0   conform/exprtest/uminus.out
     162     162       0       0   conform/exprtest/xor.out
     324     324       0       0   conform/exprtest/xoreq.out
   16291   23372       4      20   TOTAL
```

```
$$CPU$$=DF(__MOD166__)
$$FLOAT$$=FLOAT64
$$MODEL$$=LA
$$OT$$=DF(__OPT__)
BACKSLASH=\
C166INC=C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC
;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC
C166LIB=C:\KEIL\C166\LIB
C251INC=C:\C251\INC
C251LIB=C:\C251\LIB
C51INC=C:\C51\INC
C51LIB=C:\C51\LIB
CFLAGS=DB SB NOPR LA FLOAT64 DF(__MOD166__) DF(__OPT__)
COMPUTERNAME=RK
ComSpec=C:\WINNT\system32\cmd.exe
DIRNAME=xxxxxxx\xxxxxxxx\xxxxxxxxx
DPPUSE=N
EXE=.66
EXEHO=.exe
EXENAME=xxxxxxxx
HOCFLAGS=-c -AL -Za
HOLFLAGS=-F 2800
HOMEDRIVE=C:
HOMEPATH=\
LFLAGS=CLASSES (FCODE(0x10000-0x2FFFF)) CASE LINES SYMBOLS PRINT(NUL)
LOGONSERVER=\\RK
NUMBER_OF_PROCESSORS=1
OBJ=.obj
OBJHO=.obj
OS=Windows_NT
Os2LibPath=C:\WINNT\system32\os2\dll;
Path=\plumhall\test.166;C:\WINNT\system32;C:\WINNT;C:\KEIL\C166\BIN;C:\UTIL
PATHEXT=.COM;.EXE;.BAT;.CMD
PHCC=C166
PHDST=\plumhall\test.166
PHDSTDRV=D:
PHHOCC=cl
PHIN=C166INC
PHIP=D:\KEIL\C166\INC
```

```
PHLI=L166
PHSRC=\plumhall\suite
PHTMP=This is a very long string that sets aside environment space.
PHTMP2=This is a very long string that sets aside environment space.
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 5 Stepping 2, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0502
PROMPT=$P$G
STARTUP=strt66lm
SYSPATH=C:\WINNT\system32;C:\WINNT;C:\KEIL\C166\BIN;C:\UTIL
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\TEMP
TMP=C:\TEMP
USERDOMAIN=RK
USERNAME=Administrator
USERPROFILE=C:\WINNT\Profiles\Administrator
windir=C:\WINNT
WRKDRV=D:
```

9.1.3 Summary File 2

```
EXPECTED  ACTUAL  ERRORS SKIPPED  FILE NAME     5.00  1994-01-03
     16      16       0       0   conform/environ.out
     24      24       0       4   conform/interp91.out
   1224    1223       1       0 **conform/lang.out
    123    7208       0      16 **conform/libfrst.out
    821     821       0       0   conform/prec1.out
   1180    1180       0       0   conform/prec2.out
      1       1       0       0   conform/capacity/capacity.out
     11      11       0       0   conform/errauto/i91.out
    151     151       0       0   conform/errauto/j91.out
     32      30       2       0 **conform/errauto/m61.out
      0       0       0       0   conform/errauto/m62.out
     75      75       0       0   conform/errauto/m63.out
      5       4       1       0 **conform/errauto/m64.out
     77      76       1       0 **conform/errauto/m65.out
     24      24       0       0   conform/errauto/m66.out
     17      17       0       0   conform/errauto/m67.out
     39      39       0       0   conform/errauto/m68.out
    288     288       0       0   conform/exprtest/andif.out
    576     576       0       0   conform/exprtest/assign.out
    162     162       0       0   conform/exprtest/band.out
    324     324       0       0   conform/exprtest/bandeq.out
    288     288       0       0   conform/exprtest/cast.out
    162     162       0       0   conform/exprtest/compl.out
    288     288       0       0   conform/exprtest/div.out
    120     120       0       0   conform/exprtest/diveq1.out
    156     156       0       0   conform/exprtest/diveq2.out
    156     156       0       0   conform/exprtest/diveq3.out
    144     144       0       0   conform/exprtest/diveq4.out
    288     288       0       0   conform/exprtest/eq.out
    288     288       0       0   conform/exprtest/ge.out
    288     288       0       0   conform/exprtest/gt.out
     13      13       0       0   conform/exprtest/int1.out
      9       9       0       0   conform/exprtest/int2.out
      9       9       0       0   conform/exprtest/int3.out
     12      12       0       0   conform/exprtest/int4.out
     13      13       0       0   conform/exprtest/int5.out
     12      12       0       0   conform/exprtest/int6.out
     13      13       0       0   conform/exprtest/int7.out
     15      15       0       0   conform/exprtest/int8.out
     13      13       0       0   conform/exprtest/int9.out
     10      10       0       0   conform/exprtest/int10.out
     13      13       0       0   conform/exprtest/int11.out
     14      14       0       0   conform/exprtest/int12.out
     11      11       0       0   conform/exprtest/int13.out
    288     288       0       0   conform/exprtest/le.out
    162     162       0       0   conform/exprtest/lsh.out
    324     324       0       0   conform/exprtest/lsheq.out
    288     288       0       0   conform/exprtest/lt.out
    122     122       0       0   conform/exprtest/mineq1.out
    154     154       0       0   conform/exprtest/mineq2.out
    152     152       0       0   conform/exprtest/mineq3.out
    136     136       0       0   conform/exprtest/mineq4.out
    288     288       0       0   conform/exprtest/minus.out
     12      12       0       0   conform/exprtest/mix1.out
      9       9       0       0   conform/exprtest/mix2.out
      9       9       0       0   conform/exprtest/mix3.out
     12      12       0       0   conform/exprtest/mix4.out
      9       9       0       0   conform/exprtest/mix5.out
     14      14       0       0   conform/exprtest/mix6.out
     11      11       0       0   conform/exprtest/mix7.out
     17      17       0       0   conform/exprtest/mix8.out
     11      11       0       0   conform/exprtest/mix9.out
     12      12       0       0   conform/exprtest/mix10.out
     15      15       0       0   conform/exprtest/mix11.out
     16      16       0       0   conform/exprtest/mix12.out
     14      14       0       0   conform/exprtest/mix13.out
    288     288       0       0   conform/exprtest/ne.out
    288     288       0       0   conform/exprtest/not.out
    162     162       0       0   conform/exprtest/or.out
    288     288       0       0   conform/exprtest/orelse.out
    324     324       0       0   conform/exprtest/oreq.out
    122     122       0       0   conform/exprtest/plueq1.out
    156     156       0       0   conform/exprtest/plueq2.out
    154     154       0       0   conform/exprtest/plueq3.out
    144     144       0       0   conform/exprtest/plueq4.out
    288     288       0       0   conform/exprtest/plus.out
    576     576       0       0   conform/exprtest/postdec.out
    576     576       0       0   conform/exprtest/preinc.out
    576     576       0       0   conform/exprtest/quest.out
     12      12       0       0   conform/exprtest/real1.out
     11      11       0       0   conform/exprtest/real2.out
     13      13       0       0   conform/exprtest/real3.out
     11      11       0       0   conform/exprtest/real4.out
     10      10       0       0   conform/exprtest/real5.out
     12      12       0       0   conform/exprtest/real6.out
     15      15       0       0   conform/exprtest/real7.out
     11      11       0       0   conform/exprtest/real8.out
      8       8       0       0   conform/exprtest/real9.out
     15      15       0       0   conform/exprtest/real10.out
     10      10       0       0   conform/exprtest/real11.out
     16      16       0       0   conform/exprtest/real12.out
     15      15       0       0   conform/exprtest/real13.out
    162     162       0       0   conform/exprtest/remain.out
    324     324       0       0   conform/exprtest/remeq.out
    162     162       0       0   conform/exprtest/rsh.out
    324     324       0       0   conform/exprtest/rsheq.out
    120     120       0       0   conform/exprtest/timeq1.out
    154     154       0       0   conform/exprtest/timeq2.out
    156     156       0       0   conform/exprtest/timeq3.out
    146     146       0       0   conform/exprtest/timeq4.out
    288     288       0       0   conform/exprtest/timesop.out
    288     288       0       0   conform/exprtest/uminus.out
    162     162       0       0   conform/exprtest/xor.out
    324     324       0       0   conform/exprtest/xoreq.out
  16291   23371       5      20   TOTAL
```

```
$$CPU$$=MOD167
$$FLOAT$$=FLOAT64
$$MODEL$$=MD
$$OT$$=DF(__OPT__)
BACKSLASH=\
C166INC=C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC
;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC;C:\KEIL\C166\INC
C166LIB=C:\KEIL\C166\LIB
C251INC=C:\C251\INC
C251LIB=C:\C251\LIB
C51INC=C:\C51\INC
C51LIB=C:\C51\LIB
CFLAGS=DB SB NOPR MD FLOAT64 MOD167 DF(__OPT__)
COMPUTERNAME=RK
ComSpec=C:\WINNT\system32\cmd.exe
DIRNAME=xxxxxxx\xxxxxxxx\xxxxxxxxx
DPPUSE=N
EXE=.66
EXEHO=.exe
EXENAME=xxxxxxxx
HOCFLAGS=-c -AL -Za
HOLFLAGS=-F 2800
HOMEDRIVE=C:
HOMEPATH=\
LFLAGS=CLASSES (FCODE(0x10000-0x2FFFF)) CASE LINES SYMBOLS PRINT(NUL)
LOGONSERVER=\\RK
NUMBER_OF_PROCESSORS=1
OBJ=.obj
OBJHO=.obj
OS=Windows_NT
Os2LibPath=C:\WINNT\system32\os2\dll;
Path=\plumhall\test.166;C:\WINNT\system32;C:\WINNT;C:\KEIL\C166\BIN;C:\UTIL
PATHEXT=.COM;.EXE;.BAT;.CMD
PHCC=C166
PHDST=\plumhall\test.166
PHDSTDRV=D:
PHHOCC=cl
PHIN=C166INC
PHIP=D:\KEIL\C166\INC
```

```
PHLI=L166
PHSRC=\plumhall\suite
PHTMP=This is a very long string that sets aside environment space.
PHTMP2=This is a very long string that sets aside environment space.
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 5 Stepping 2, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0502
PROMPT=$P$G
STARTUP=strt67mm
SYSPATH=C:\WINNT\system32;C:\WINNT;C:\KEIL\C166\BIN;C:\UTIL
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\TEMP
TMP=C:\TEMP
USERDOMAIN=RK
USERNAME=Administrator
USERPROFILE=C:\WINNT\Profiles\Administrator
windir=C:\WINNT
WRKDRV=D:
```

9.1.4 Explanation Of Error Reports From The Plum Hall Test

The PLUM HALL test reports 4 Errors for the Keil C166 compiler that are characterised below:

9.1.4.1 LANG Problem

LANG: this is a complex macro that is not expanded correctly:

```
ERROR in c68.c at line 239: "1+2+h" != "NEITHER '1+f(2)' NOR '1+h(2)'"
```

This problem appears when multiple nested macros are called with text expansions.

9.1.4.2 M61 Problem

M61: there is not compiler error message for following construct:

```
enum { A = INT_MAX, B };  // B overflows 'int' value range for enum
```

9.1.4.3 M64 Problem

M64: there is not compiler error message for:

```
     switch (i)   {
        case LONG_MAX * 4 :     // LONG_MAX * 4 overflows value range
```

9.1.4.4 M65 Problem

M65: is a duplication of the problem M61:

```
          int main() {
     enum { A = INT_MAX, B };  // Missing Error message
          return 0; }
```

10 Appendix G

10.1 Known Problems In C166 v3.12j

(These problems have been fixed in C166 v3.12k)

```
33) INCDIR: increased to max. 50 path-specs.
============================================
CORR: BOM.H (/5.10.98/)

34) Assembly-Errors when SRC is used
====================================
#pragma ot(3,size)
#pragma hold(near 2048)

extern unsigned char m[1034];

struct x {
  int x[5];
};
int i;

typedef struct {
  unsigned char      s;
  char o;
} A;
const static A far a [] = {
  { 1, 2 }, { 3, 4 },
};
const static A near b [] = {
  { 1, 2 }, { 3, 4 },
};
void main (void)  {
  ((struct x *) m)->x[b[0].o] = 0;
  ((struct x *) m)->x[a[0].o] = 0;
  ((struct x *) m)->x[i] = 0;
}
CORR: REG166.C, ASMGEN.C (/9.10.98/)

35) Gp or hangup
================
CORR: CFOLD.C (PostFold /9.10.98/)

36) tagName space clash
=======================
struct test_struct {           // outer 'test_struct'
      int structivar0;
      int structivar1;
};
int   z;

void main (void)  {
  struct test_struct {      // inner 'test_struct'
    int i;
    struct test_struct *next;  // error: resolves to outer 'test_struct'
  };
  struct test_struct *ptr; // , array[15];
  if ( ptr->next->i != 25) z = 1;    // gives 'next' undefined member
}
CORR: GRM.C (rsutp /22.10.98/)

37) Incorrect Warning 'expr with no effect' and code removal
============================================================
much like #30 !
  ((*(ANWEISUNGtyp *)memcpy (&gv0.BUF_AB[PROG_LES_BUFinst.IX],
                             &gv0.SATZ[PROG_LES_BUFinst.LOOPCOUNTER],
                             sizeof(gv0.BUF_AB[PROG_LES_BUFinst.IX]))));
CORR: GENEXP.C (GenExp /3.11.98/)
```

```
38) Missing Error Meassage on constant modification
===================================================
const int c_val = 3;
void main (void)  {
  c_val = c_val + 2;  // gives correct error meassage
  c_val +=  2;        // is not considered as a constant modification
}
CORR: ENODE.C (DoAdd /25.2.99/)

39) Unnecessary Warning
=======================
static unsigned char suc;
char test (void)  {
  register signed char rsc;
  static   signed char ssc;

  if ((unsigned char)rsc >= suc)       // *** WARNING 173 IN LINE 9 OF TEST130.C: '>=':
signed/unsigned type mismatch
  rsc++;
  if ((unsigned char)ssc >= suc)
  ssc++;
  return (ssc+rsc);
}
CORR: CFOLD.C (RelOp8 /25.2.99/)


40) Word access to a odd address when pragma order is used
==========================================================
#pragma code debug or

void test(char *b, char *c) {;}

char const abc = 3;

void main (void) {
  char dummy2[] = "hello";  // 'hello' starts at an odd address and is copied word by word
  char dummy3[] = "hell";

  test(dummy2, dummy3);    // use dummy variables
}
```