OS-9 C COMPILER
VERSION 3.2
RELEASE NOTES

ACKNOWLEDGEMENTS

Many thanks to Richard Russell, Larry Crane, and Dave West.

COPYRIGHT AND REVISION HISTORY

This manual reflects Version 2.3 of the OS-9 Operating System and Version 3.2 of the OS-9 C Compiler.

Publication Editor: Kathleen Flood
Publication Date: March 1990
Product Number: ccc68na68rn

DISCLAIMER

REPRODUCTION NOTICE

TRADEMARKS

Microware Systems Corporation • 1900 N.W. 114th Street
Des Moines, Iowa 50322 • Phone: 515/224-1929

Table of Contents

Documentation Corrections

Known Bugs

Edition Numbers

End of Table of Contents

Introduction

Package Contents

The OS-9 C Compiler Version 3.2 distribution package contains the following items:

- 1 or more product disks (product number ccc68na68ei)
- ***OS-9 C Compiler Version 3.2 Release Notes*** (this manual)

New customers will also receive these manuals:

- ***OS-9 C Compiler User's Manual***
- ***OS-9 Assembler/Linker Manual User's Manual***
- ***OS-9 Debugger User's Manual***

Distribution Changes

Prior to January 1990, the 68000 and 68020/030 C compilers were sold separately. Microware now distributes one compiler package which contains both of these compilers. End users who purchased version 3.1 or earlier may contact their OEM or Microware (if the package was purchased directly) for 68020/30 upgrades.

Two new header files are included with this release: `limits.h` and `float.h`. `Limits.h` contains the maximum and minimum values for different data types. `Float.h` contains the maximum and minimum values for data types specific to floating point math.

The following header files have been updated for this release: `direct.h`, `errno.h`, `math.h`, `module.h`, `procid.h`, `rbf.h`, `sbf.h`, `setsys.h`, `sgcodes.h`, `sg_stat.h`, and `signal.h`.

New versions of `cio` and `cio020` are included in this release. A "`trap handler mismatch`" error will occur if any attempt is made to run the new compiler with the old `cio` or `cio020`.

ANSI Compatibility

The following C functions are ANSI-compatible:

asctime	atof	atoi	atol	clock	ctime
difftime	frexp	gmtime	ldexp	localtime	memchar
memcmp	memcpy	memmove	memset	mktime	printf
sprintf	strtod	strtol	strtoul	system	time

Manual Overview

The ***OS-9 C Compiler Version 3.2 Release Notes*** contain descriptions of all changes to Microware's C compiler, assembler, and linker since the 2.3 release of OS-9. The release notes are divided into the following sections:

- ***New C Functions***
  This section describes new C functions included in the Version 3.2 Release.
- ***Enhancements and Corrected Bugs***
  This section describes changes in the software since the last release.
- ***Documentation Corrections***
  This section lists corrections to existing C compiler documentation.
- ***Known Bugs***
  This section describes all known bugs in the Version 3.2 software.
- ***Edition Numbers***
  This section contains a list of current edition numbers.

New C Functions

Although the OS-9/68000 C Compiler Version 3.2 release is primarily a "maintenance" release, twelve new C functions have been added. Their descriptions are included in this section:

- **_atou**
- **_lcalloc**
- **_lfree**
- **_lmalloc**
- **_lrealloc**
- **_mallocmin**
- **frexp**
- **ldexp**
- **memmove**
- **strtod**
- **strtol**
- **strtoul**

_atou() Alpha to Unsigned Conversion

SYNOPSIS:

```
unsigned _atou(string)
char     *string;
```

FUNCTION: `_atou()` converts a string into its appropriate unsigned numeric value, if possible. `string` points to a string that contains a printable representation of a number expressed in the following format: `[+/-]<digits>`. `_atou()` treats long and int values identically.

_lcalloc() Allocate Storage for Array (low-overhead)

SYNOPSIS:

```
void *_lcalloc(nel,elsize)
unsigned long  nel,        /* number of elements in array */
               elsize;     /* size of elements */
```

FUNCTION: This function allocates space for an array. `nel` is the number of elements in the array, and `elsize` is the size of each element. The allocated memory is cleared to zeroes.

This function calls `_lmalloc()` to allocate memory. If the allocation is successful, `_lcalloc()` returns a pointer to the area. If the allocation fails, `_lcalloc()` returns zero (NULL).

NOTE: Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, `realloc()`) saves eight bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

CAVEATS: Extreme care should be used to insure that only the memory assigned is accessed. Modifying addresses immediately above or below the assigned memory will cause unpredictable program results.

The low-overhead functions require that the *programmer* keep track of the sizes of allocated spaces in memory. (Note the `_lfree()` and `_lrealloc()` parameters.)

SEE ALSO: `_srqmem()`, `_srtmem()`, `_lfree()`, `_lmalloc()`, `_lrealloc()`

_lfree() Return Memory (low-overhead)

SYNOPSIS:

```
void _lfree(ptr, size)
void            *ptr;   /* pointer to memory to be returned */
unsigned long   size;   /* size of memory to be returned */
```

FUNCTION: `_lfree()` returns a block of memory granted by `_lcalloc()` or `_lmalloc()`. The memory is returned to a pool of memory for later re-use by `_lcalloc()` or `_lmalloc()`.

`_lfree()` never returns an error.

NOTE: Use of the low-overhead allocation functions (`_lcalloc()`, `lmalloc()`, `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, `realloc()`) saves eight bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

CAVEATS: If `_lfree()` is used with something other than a pointer returned by `_lmalloc()` or `_lcalloc()`, the memory lists maintained by `_lmalloc()` will be corrupted and programs may behave unpredictably.

The low-overhead functions require that the *programmer* keep track of the sizes of allocated spaces in memory.

SEE ALSO: `_lcalloc()`, `_lmalloc()`, `_lrealloc()`

_lmalloc() Allocate Memory from an Arena (low-overhead)

SYNOPSIS:

```
void *_lmalloc(size)
unsigned long   size;   /* size of memory block to allocate */
```

FUNCTION: `_lmalloc()` returns a pointer to a block of memory of `size` bytes. The pointer is suitably aligned for storage of data of any type.

`_lmalloc()` maintains an amount of memory called an ***arena*** from which it grants memory requests. `_lmalloc()` will search its arena for a block of free memory large enough for the request and, in the process, coalesce adjacent blocks of free space returned by the `_lfree()` function. If sufficient memory is not available in the arena, `_lmalloc()` calls `_srqmem()` to get more memory from the system.

`_lmalloc()` returns zero (NULL) if there is no available memory or if the arena is detected to be corrupted.

NOTE: Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, `realloc()`) saves eight bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

CAVEATS: Extreme care should be used to insure that only the memory assigned by `_lmalloc()` is accessed. Modifying addresses immediately above or below the assigned memory or passing `_lfree()` a value not assigned by `_lmalloc()` will cause unpredictable program results.

The low-overhead functions require that the *programmer* keep track of the sizes of allocated spaces in memory. (Note the `_lfree()` and `_lrealloc()` parameters.)

SEE ALSO: `_lcalloc()`, `_lfree()`, `_lrealloc()`

_lrealloc() Resize a Block of Memory (low-overhead)

SYNOPSIS:

```
void *_lrealloc(oldptr, newsize, oldsize)
void           *oldptr;   /* old pointer to block of memory */
unsigned long  newsize;   /* size of new memory block */
               oldsize;   /* size of old memory block */
```

FUNCTION: `_lrealloc()` re-sizes a block of memory pointed to by `oldptr`. `oldptr` should be a value returned by a previous `_lmalloc()`, `_lcalloc()` or `_lrealloc()`.

`_lrealloc()` returns a pointer to a new block of memory. The size of this new block is specified by `newsize`. The pointer is aligned to store data of any type.

If `newsize` is smaller than `oldsize`, the contents of the old block are truncated and placed in the new block. Otherwise, the entirety of the old block's contents begin the new block.

The results of `_lrealloc(NULL,newsize,0)` and `_lmalloc(size)` are the same.

`_lrealloc()` returns zero (NULL) if the requested memory is not available or `newsize` is specified as zero.

NOTE: Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, `realloc()`) saves eight bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

CAVEAT: The low-overhead functions require that the *programmer* keep track of the sizes of allocated spaces in memory.

SEE ALSO: `_freemin()`, `_lcalloc()`, `_lfree()`, `_lmalloc()`

_mallocmin() Set Minimum Allocation Size

SYNOPSIS:

```
_mallocmin(size)
unsigned size;          /* minimum allocation size in bytes */
```

FUNCTION: `_mallocmin()` sets the minimum amount of memory that allocation functions may request through `srqmem()`. The `size` parameter cannot be less than the system memory block size. If a smaller size is requested, `size` is automatically set to the system memory block size.

OS-9 allows each process only 32 different memory segments; therefore, `size` should be increased if a program requires a great amount of memory. The extra space may be necessary if memory is fragmented.

`_mallocmin()` never returns an error.

SEE ALSO: `_lcalloc()`, `_lmalloc()`, `_lrealloc()`, `calloc()`, `malloc()`, `realloc()`

frexp() Returns Parts of a Floating Point Number

SYNOPSIS:

```
double frexp(value,exp)
double value;           /* floating point value */
int    *exp;            /* exponent */
```

FUNCTION: `frexp()` breaks a floating point `value` into a normalized fraction and an integral exponent of two (`exp`). This function returns the fraction `x`, such that `1/2 <= x < 1` and `value = x * 2^exp`.

ldexp() Multiply Float by Exponent of 2

SYNOPSIS:

```
double ldexp(fp,exp)
double fp;              /* floating point value */
int    exp;             /* exponent */
```

FUNCTION: `ldexp()` multiplies a floating point value by an integral power of two. This function returns the value equal to `fp * 2^exp`.

memmove() — Move Memory

SYNOPSIS:

```
#include <strings.h>

memmove(dest, src, size)
char     *dest,          /* pointer to destination memory */
         *src;           /* pointer to memory to copy */
unsigned size;           /* size of memory to copy */
```

FUNCTION: `memmove()` copies a specific number of bytes from the region specified by `src` to the region specified by `dest`. The number of bytes copied is specified by `size`. Regions may overlap with no ill effects.

strtod() — String to Double Conversion

SYNOPSIS:

```
#include <math.h>

double strtod(nptr,endptr)
char   *nptr;    /* pointer to beginning of string */
char   **endptr; /* specifies first character after converted string */
```

FUNCTION: `strtod()` parses a character string and converts it to the associated numeric value of type `double`. The beginning of the string is pointed to by `nptr`. All leading white space is ignored by `strtod()`.

If successful, `strtod()` returns the converted value. The address of the first character past the converted string is pointed to by `endptr`, if `endptr != (char **) NULL`.

If the string to be converted is empty, or the string does not have the appropriate form, `strtod()` returns zero (NULL) and stores `nptr` in `*endptr`.

If the converted value causes an overflow, `strtod()` returns `HUGE_VAL`, with the appropriate preceding sign. If the converted value causes an underflow, `strtod()` returns zero. In either of these two cases, `ERANGE` is placed in `errno`.

SEE ALSO: `strtol()`, `strtoul()`

strtol(), strtoul() String to Long Conversion

SYNOPSIS:

```
#include <limits.h>

unsigned long strtol(nptr, endptr, base)
char *nptr;      /* pointer to beginning of string */
char **endptr;   /* specifies first character after converted string */
int  base;       /* base of number string */

unsigned long strtoul(nptr, endptr, base)
char *nptr;      /* pointer to beginning of string */
char **endptr;   /* specifies first character after converted string */
int  base;       /* base of number string */
```

FUNCTION: `strtol()` parses a character string and converts it to the associated numeric value of type `long`. `strtoul()` converts the string into an `unsigned long` value. In all other respects, `strtoul()` and `strtol()` are identical.

The beginning of the string is pointed to by `nptr`. `strtol()` ignores all leading white space.

`base` is the base of the number string to be converted. For example, `strtol("377", NULL, 8)` will return 255. `base` must be zero or in the range of 2 - 36. If `base` is 16, `strtol` will accept a leading `0x` or `0X`. "Digits" from 10 through 35 should be represented by the letters 'a' through 'z' respectively (case is not significant). Only digits valid with the specified base are parsed.

If a `base` of zero is specified, then `strtol()` will accept any form of constant that the C compiler will accept, excluding `l` or `L`, which as a suffix indicates a long constant. Optional preceding signs are also accepted. For example, `strtol("0xffff", NULL, 0)` returns 65535.

If the string to be converted is empty or does not have the appropriate form, `strtol()` will return zero (NULL) and `*endptr` will contain `nptr`.

If the converted value causes an overflow, `strtol()` will return `LONG_MAX` or `LONG_MIN` to indicate positive or negative values, respectively. If the converted value causes an underflow, `strtol()` will return zero. In either of these cases, `ERANGE` is placed in `errno`.

SEE ALSO: `strtod()`

End of New C Functions

Enhancements and Corrected Bugs

This section contains descriptions of all enhancements and corrected problems in Microware's C compiler, assembler, and linker software since the OS-9 Version 2.3 Release.

C Compiler Executive (cc):

The compiler executive now passes the following pre-defined names to the preprocessor. The exact combinations defined by the compiler executive are determined by the `-to` and `-tp` options.

Microprocessor Family: `_MPF68K`
`_MPF68020`
`_MPF386`

Floating Point Processor Family: `_FPF68881`
`_FPF80387`

Byte Ordering: `_BIG_END` (for big endian machines)
`_LIL_END` (for little endian machines)

Eight options have been added to the compiler executive:

`-cs=<root psect>` Specifies an alternate root psect. The path name is considered relative to the current data directory. By default, the root psect is `cstart.r`.

`-lo=<linker options>` Passes the specified options to the linker. Only white space is recognized as a delimiter, because the linker does not support options that require quoted strings. **NOTE:** Wild-card characters (`?` or `*`) cannot be used in the linker options.

`-nl` Prevents the use of default libraries during linking.

`-nv` Forces the compiler to suppress vsect directives from the code it generates. This option is used to make OS-9000 descriptors. **NOTE:** The `-nv` option is only valid for OS-9000 targets.

`-o[=<level>]` Sets the optimization level. Optimization shortens object code and increases execution speed; it is recommended for production versions of debugged programs. `<level>` may have the following values:

- 0 no optimization
- 1 Branch chains are collapsed and common tails of basic blocks with the same destination are merged.
- 2 The optimization of (1) is performed, and some superfluous memory accesses are avoided.

If no level is specified, no optimization is performed. If this option is not specified, level 1 optimization is performed.

`-t?` Lists target processor specific options.

`-to=<name>` Specifies the target operating system by name:

`osk`	OS-9/68000 Operating System
`os9k`	OS-9000 Operating System
`os9000`	OS-9000 Operating System

`-tp=<n>[<tp_opts>]` Specifies the target processor `<n>` and the target processor options to be used. The possible values for `<n>` include:

`68k`	MC68000, MC68010, MC68070
`68000`	MC68000, MC68010, MC68070
`68020`	MC68020, MC68030
`020`	MC68020, MC68030
`80386`	I80386
`386`	I80386

The possible values for `<tp_opts>` include:

`DW`	word data access (68k default)
`DL`	long word data access (68020 default)
`CW`	word code access (68k default)
`CL`	long word code access (68020 default)
`I`	does not emit 68881 instructions
`J`	prevents linker from creating jumptable

NOTE: If `-i` is not specified, in-line floating point is the default for 68020 target processors.

C Preprocessor (cpp):

The preprocessor no longer destroys user memory when passed a line exactly 512 characters long.

The preprocessor no longer uses fixed-size buffers for input and macro expansion. This eliminates bus errors caused by accessing memory beyond the fixed-size buffer.

All macros are now expanded properly. Previously, in rare instances, the preprocessor failed to expand macros that should have been expanded.

The preprocessor no longer uses fixed-size buffers for input text and expanded text. This allows code with large macros to be preprocessed. Miscellaneous changes have also made the input routines more efficient.

C Compiler (c68/c68020):

The compiler no longer overwrites source-level debugging information for typedef structures which contain typedef structures.

The compiler now generates correct code for structure assignments of the form `*p1++ = *p2++`, where p1 and p2 are in registers. Previously, the size of p1 or p2 was confused with the size of the units in which memory was moved for the assignment.

The compiler now generates correct code for all structure assignments. Previously, structure assignments moved too much data if the structure did not require alignment or if the source and destination were the result of de-referencing post-incremented register variables.

The compiler now generates correct code if a program attempts (via casts) to use a pointer to a function as if it was an array.

The compiler now generates correct code for post-increments of variables cast to a narrower type.

The compiler no longer aborts (exit status 139) when a source file contains a string longer than 512 characters or an unmatched quote/parenthesis.

The compiler now generates correct code for statements consisting solely of `?:` expressions in which the second operand is a post-incremented/decremented variable.

The compiler now generates correct code for *= expressions whose left-hand side is a char or an unsigned char.

The compiler no longer requires the #line directive to be followed by file name.

The compiler no longer loses any of the four least significant bits when loading certain bit fields into a register.

The compiler no longer uses fixed-size buffers for input text. Miscellaneous changes have also made the input routines more efficient.

The compiler now performs some checking to avoid superfluous variable loads from memory.

The compiler now generates improved code for some bit field operations, i.e. comparing single-bit fields with zero or determining whether signed bit fields are negative.

The compiler no longer returns an error if the last brace in a program is followed by a semi-colon.

The libraries have been re-grouped so that the following error does not occur during compilation: "Symbol '_T$Mul' defined in psect 'Longs' in file '/dd/lib/math881.l' has already appeared in psect 'cmul_a' in file '/dd/lib/clib020h.l'." The same error previously occurred with labels _T$LMul and _T$UDiv, as well.

68020 C Compiler only:

The compiler now generates correct code for return <expression> statements. Previously, the compiler generated incorrect code if this statement appeared in a function returning a float value into a 68881/2 register.

SUN3/4 Cross C Compiler only:

The compiler no longer aborts (exit status 139) when initializing an undeclared structure.

68881 code only:

Functions expected to return float no longer return a double in d0/d1 if the expression being returned is evaluated into a 68881 register.

Assembler (r68):

VAX/VMS Cross Assembler:

r68 now assembles rept statements correctly.

Linker (l68):

There is no longer a limit on the number of times the -l=<path> option can be repeated in a command line. Previously, -l could only be repeated 32 times per command line.

The <path> specification of the -o=<path> option is relative to the current execution directory. If <path> is not found relative to the current execution directory, the linker searches relative to the current data directory.

The -z and -z=<file> options now recognize command line options. For example, if the contents of a file named linklist are as listed below, then command lines ① and ② are equivalent:

```
file1.r
file2.r
file3.r
-l=lib1
-l=lib2
-gM=4k
-o=jive

① l68 -z=linklist

② l68 file1.r file2.r file3.r -l=lib1 -l=lib2 -gM=4k -o=jive
```

OS-9 C Library

The `atol()`, `memset()`, and `system()` functions now conform to the ANSI standard, except when used with unsupported types (i.e. long double).

Code supporting `malloc()/free()/realloc()` has been changed so that `ebrk()` and `malloc()` calls in the same program do not confuse `malloc()`.

The `localtime()` function now adjusts to local time before checking for Daylight Savings Time (DST). `localtime()` also switches to DST correctly.

The `printf()` and `sprintf()` functions now conform to the ANSI standard, except when used with unsupported types (i.e. long double). Most notably, an asterisk (*) can now be used as a field width or precision to notify `printf()` to use an argument (int) for the field width and/or precision.

The `memset()` function is significantly faster; therefore, `calloc()` is also faster.

The `_atou()`, `_lcalloc()`, `_lfree()`, `_lmalloc()`, `_lrealloc()`, `_mallocmin()`, `frexp()`, `ldexp()`, `memmove()`, `strtod()`, `strtol()`, and `strtoul()` functions have been added to the library. Refer to the ***New C Functions*** section of these release notes for complete descriptions.

The `_mkdata_module()` function now accepts two optional parameters: type/language and memory color. These parameters follow `_mkdata_module()`'s required parameters. Bit 15 of the perm parameter is a flag which indicates the presence of optional parameters. If bit 15 is set, the optional parameters are present. If bit 15 is clear, these parameters are not present.

End of Enhancements and Corrected Bugs

Documentation Corrections

This chapter contains corrections to Microware's compiler documentation.

C Compiler User's Manual

In revision H of the manual, the tick line of the `_sysdate()` format description (page 35) should be replaced with the following:

tick:	No. of ticks/second	Current clock tick

The caveat listed in the `atoi()` description should be removed.

The `atol()` description should appear as follows:

> `atol()` converts the string specified by string into its equivalent representation in type long. `atol()` recognizes an optional sign followed by a digit string:
>
> ```
> [+/-]digits
> ```

The third line of the `memset()` synopsis should appear as follows: `void *s`

The first three sentences of the `modload()` description are incorrect and should be replaced with the following:

> `modload()` loads all modules in the file specified by the path `modname`.

The following passage should be added to the `printf()` and `sprintf()` descriptions:

> The field width and/or precision may be specified by an asterisk (*) instead of a digit string. In these cases, the field width and/or precision is specified by an int argument. These arguments must appear before the argument to be converted. A negative field width argument is interpreted as a '-' flag followed by a positive field width. A negative precision argument is ignored.

The following line should be added to the `rindex()` synopsis:

```
#include <strings.h>
```

The `system()` function description should appear as follows:

> `system()` passes its argument (`string`) to the host environment to be executed by the command processor indicated by the `SHELL` environment variable. If the `SHELL` variable is not set, the `shell` command processor is assumed. The argument string is executed as a command line by the command processor.
>
> The calling process is suspended until the `shell` command is completed. `system()` returns the exit status of the created shell.
>
> A null pointer may be used for `string` to inquire whether the command processor exists. In this case, `system()` returns a non-zero value if and only if the command processor is available. If passed a non-null pointer, `system()` returns the exit status of the command processor.

End of Documentation Corrections

Known Bugs

This section contains descriptions of all known problems in Microware's C compiler, assembler, and linker software. These problems will be corrected in subsequent releases.

C Preprocessor (cpp):

The preprocessor replaces labels within double-quoted print strings. The Reiser C preprocessor also violates K&R in this respect; therefore, existing commercial code depends on this "feature."

C Compiler (c68/c68020):

When an invalid octal value is placed in a structure, the compiler returns the error "too many elements."

There is sometimes a 32K limit on the size of the code generated. The compiler emits assembly code for string literals at the end of the code segment in which they are declared. The string literals are referenced using 16-bit PC-relative addressing. If the distance between the reference and the string literal is greater than 32K, 16-bit PC-relative references cannot reach the literal. This problem can be worked around by using the `cc` option to force 32-bit PC-relative referencing, but the program size and execution speed will suffer.

The compiler generates incorrect code for initializers that attempt to initialize a variable to the offset of a field within a structure. Problem initializers generally have the following form: `int i = (int) &((struct woof*)0) -> field.field2`

68020 C Compiler only:

The compiler returns an "out of range" error on a `move.l` instruction when more than 32K of automatic variables exists in a single function.

Assembler

r68 (68000 Macro Assembler):

The functions of the assembler `-c` and `--c` options are reversed. Currently, these options operate as follows:

`r68 -c`	Conditional assembly listing
`r60 --c`	No conditional assembly listing

The assembler does not check `tas` statement syntax accurately.

If `r68` assembles a program that contains an `opt l` directive followed by `opt -l`, then `opt -l` appears in the listing.

`r68` does not assemble the `cas2` 68020 instruction correctly.

`r68` returns a "line too long" error when assembling a file whose last line is not terminated by a standard end-of-line character.

r68020 (68020 Macro Assembler):

When `r68020` is in 68030 mode, it does not flag the `CALLM` and `RTM` instructions as "68020-only".

The `-m=<num>` option prints the version/edition number and ignores everything on the command line past the equal sign. The command line is processed correctly if the `-m` option is used without the equal sign.

`r68020` does not assemble the `cas2` instruction correctly.

Linker

168 (68000 Linker):

The linker does not detect an error when the sign of a byte offset changes due to relocation. For example, the following SCF driver code fragment correctly assembles without error:

```
00141                       vsect
  .
  .
  .
00151    00000060     WakeUpQue    ds.w    NUMCHANS
  .
  .
  .
00209                       ends
  .
  .
  .
00669    03f8 3580          move.w  d0,WakeUpQue(a2,d7.w)
              7060
```

However, the linker adds the SCF static area offsets when the driver is linked with `scfstat.l`. This changes the effective addresses such that a negative offset results:

```
         03f8 3580          move.w  d0,WakeUpQue(a2,d7.w)
              70b4
```

Because the linker cannot determine whether the byte being relocated is an offset or an unsigned byte constant, this is unlikely to be corrected in the future. Therefore, programmers should avoid relocatable symbolic byte offsets in indexed addressing modes.

End of Known Bugs

Notes

Edition Numbers

This section contains list of current edition numbers.

Version 3.2 Resident C Compiler Edition Numbers

Module Name	File Name	Edition	CRC
cc	cc	44	0339B5
cpp	cpp	41	30C434
c68	c68	357	1BDF6E
c68020	c68020	357	4B998B
o68	o68	19	0E66EF
r68	r68	67	DF2223
r68020	r68020	95	2DA5A4
168	168	64	A3F33B
cio	cio	6	BD5171
cio020	cio	6	9288AB

End of Edition Numbers

Notes