



Intel[®] IXP2400/IXP2800 Network Processors

Microengine C Compiler LibC Library Reference Manual

November 2003

Revision History

Date	Revision	Description
January 2002	001	Pre-Release 2
May 2002	002	Release for the IXA SDK 3.0
August 2002	003	Release for the IXA SDK 3.0 Pre Release 4
November 2002	004	Release for the IXA SDK 3.0 Pre Release 5
January 2003	005	Release for the IXA SDK 3.0 Pre Release 6
September 2003	006	Release for the IXA SDK 3.5
November 2003	007	Release for the IXA SDK 3.5

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The software described in this User's Guide may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

Intel SpeedStep, Intel Thread Checker, Celeron, Dialogic, i386, i486, iCOMP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetStructure, Intel Xeon, Intel XScale, Itanium, MMX, MMX logo, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation 2002–2003.

Contents

1	LibC Functions	5
1.1	About This Book	5
1.2	String Literals	5
1.3	stdlib.h	6
1.4	ixptypes.h.....	7
1.4.1	Basic types	7
1.4.2	Memory region types	7
1.5	memory.h.....	7
1.5.1	memcpy().....	8
1.5.2	memmove().....	9
1.5.3	memset().....	12
1.5.4	memcmp().....	13
1.5.5	memchr()	15
1.6	string.h.....	16
1.6.1	strlen().....	16
1.6.2	strcmp().....	17
1.6.3	strncmp().....	18
1.6.4	strcpy().....	19
1.6.5	strncpy().....	20
1.6.6	strcat().....	21
1.6.7	strncat().....	22
1.6.8	strchr()	23
1.6.9	strrchr()	24
1.6.10	strstr()	25
1.6.11	strspn().....	26
1.6.12	strcspn().....	27
1.6.13	strpbrk()	28
1.6.14	strtok().....	29
1.6.15	strtol().....	30
1.6.16	strtoul().....	31

Tables

1	memcpy() Prototypes.....	8
2	memmove() Prototypes	10
3	memset() Prototypes	12
4	memcmp() Prototypes	13
5	memchr() Prototypes	15
6	strlen() Prototypes	16
7	strcmp() Prototypes	17
8	strncmp() Prototypes	18
9	strcpy() Prototypes.....	19
10	strncpy() Prototypes.....	20
11	strcat() Prototypes	21
12	strncat() Prototypes	22
13	strchr() Prototypes	23
14	strrchr() Prototypes.....	24
15	strstr() Prototypes	25



16	strspn() Prototypes	26
17	strcspn() Prototypes	27
18	strpbrk() Prototypes	28
19	strtok() Prototypes	29
20	strtol() Prototypes	30
21	strtoul() Prototypes	31

1.1 About This Book

This document provides information about the functions contained in the Intel[®] Microengine C Compiler LibC library for the IXP2XXX network processors. This library provides support for a modified subset of the standard C library. Most standard library headers defined in C89/C99 are not supported at this time. Only `<memory.h>`, `<string.h>` and `<stdlib.h>` are implemented with significantly fewer functions supported than required by C89/C99.

The LibC library functions are similar to the standard C library functions with the following exceptions:

- Function names without attached memory types operate in SRAM. For example, the `memcpy()` function operates on buffers in sram while `memcpy_sram_dram()` copies a buffer from dram to sram.
- All pointers are aligned on a natural boundary according to memory region type. That is, pointers are 32 bits in sram, scratch, and local memory and 64-bits for dram. Byte, shorts, and ints are no longer aligned on a word boundary as was previously the case for the IXP1200 version of the Microengine C LibC library.
- Support for the special character types that was included in IXP1200 version of the Microengine C LibC library have been removed. The following types do not exist in this version of LibC:

- `cp_sram`
- `cp_dram`
- `cp_scratch`
- associated `cp_*`

1.2 String Literals

String literals are placed into SRAM. You should not use a string literal in a position which expects a pointer to a non-SRAM memory region, unless a static initialization of a character array is being performed. This will produce an error. For example:

```
void foo(__declspec(dram) char *str_in_dram) { ... }

foo("string"); // ERROR: "string" is in SRAM and cannot be passed to foo()

foo((__declspec(dram) char *)"string"); // RUNTIME ERROR: address of "string" is
not a valid DRAM address
{
    __declspec(dram) char *ptr = "string"; // ERROR: "ptr" must be a character
array
    __declspec(dram) char arr[7] = "string"; // CORRECT: static initialization of
character array
    foo(arr); // CORRECT: type of parameter matches
```

```
type of argument  
}
```

1.3 **stdlib.h**

This section describes the `stdlib.h` macros and functions supported by the Microengine C LibC library.

```
int abs(int n);
```

Return the absolute value of `n`.

```
long labs(long n);
```

Return absolute value of `n` (same as `abs()`).

```
int atoi(__declspec(sram) CHAR *s);  
__int64 atoi64(__declspec(sram) CHAR *s)
```

Convert string `s` to 32-bit integer or a 64-bit integer.

```
int rand();
```

Generates a random number between 0 and `RAND_MAX`. If the macro `IXP_RAND` is defined, the routine uses the IXP processor's built-in random number generator.

```
void srand(unsigned int seed);
```

Seed a new pseudo-random number sequence.

```
unsigned int _rotr(unsigned int v, int shift);  
unsigned int _rotl(unsigned int v, int shift);
```

These functions, which are non-ANSI-C compatible, rotate bits to the right or left for 32-bit ints.

```
int tolower(int c);
```

Return lowercase of character `c`.

```
int toupper(int c);
```

Return uppercase of character `c`.

```
int islower(int c);
```

Return 1 if `c` is a lower-case character, and 0 otherwise.

```
int isupper(int c);
```

Return 1 if `c` is an upper-case character, and 0 otherwise.

```
void exit(int);
```

Abort or exit the current context. For the `exit` function, the code returned is the integer argument. The `exit()` function is implementation-dependent and is defined in `rtl.c`. The current implementation kills the current thread with an `ctx_arb[kill]`.

1.4 ixtypes.h

The `ixtypes.h` file contains abbreviated type definitions for your convenience. [Section 1.4.1](#) and [Section 1.4.2](#) describe these type definitions.

1.4.1 Basic types

```

typedef int                bool;
typedef void               VOID;
typedef unsigned char     U8;
typedef char               I8;
typedef unsigned short    U16;
typedef short              I16;
typedef unsigned int      U32;
typedef int                I32;
typedef unsigned long long U64;
typedef long long          I64;
typedef char               CHAR;
typedef unsigned char     UCHAR;

```

1.4.2 Memory region types

All of the combinations of the IXP memory regions and the basic data types (with the exception of `UCHAR` and `bool`) have been defined as single types. The following are some examples of these combinations.

```

typedef __declspec(sram, CHAR)    SRAM_CHAR;
typedef __declspec(local_mem, U32) LMEM_U32;
typedef __declspec(msf_ctrl, I64) MSF_I64;
typedef __declspec(sdram, I16)    DRAM_I16;

```

1.5 memory.h

The `memory.h` file contains four memory functions that operate in the same way as the standard C memory functions with the exception of the `memchr()` function, which returns an offset to the first occurrence of a character within a buffer rather than the address of the character.

A memory function that does not contain a memory region in its name works only with buffers in SRAM. Each of the four memory functions, however, have memory-specific counterparts that work on buffers from different memory regions. For example, the `memcpy_sram_dram()` function is used to copy a buffer from DRAM to a buffer in SRAM. The function naming convention used with all memory-specific functions is to specify the destination memory region first followed by the source memory region.

The following sections describe each of the functions contained in the `memory.h` header file.

1.5.1 memcpy()

```
__declspec(sram) void *memcpy(__declspec(sram) void *dest,
                              __declspec(sram) const void *src,
                              unsigned int count);
```

Like the standard C memcpy function, this function copies up to **count** characters from memory buffer **src** into buffer **dest** and returns **dest**. Both memory buffers are in SRAM. This function correctly handles non optimal alignment cases.

Similar functions are provided for operation in all memory regions. [Table 1](#) summarizes the prototypes for each of the memory-specific memcpy() functions.

Table 1. memcpy() Prototypes (Sheet 1 of 2)

Prototype	Description
<code>__declspec(dram) void *memcpy_dram_dram(__declspec(dram) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in dram.
<code>__declspec(dram) void *memcpy_dram_lmem(__declspec(dram) void *dest, __declspec(local_mem) const void *src, unsigned int count);</code>	Copy from src in lmem to dest in dram.
<code>__declspec(dram) void *memcpy_dram_scratch(__declspec(dram) void *dest, __declspec(scratch) const void *src, unsigned int count);</code>	Copy from src in scratch to dest in dram.
<code>__declspec(dram) void *memcpy_dram_sram(__declspec(dram) void *dest, __declspec(sram) const void *src, unsigned int count);</code>	Copy from src in sram to dest in dram.
<code>__declspec(local_mem) void *memcpy_lmem_dram(__declspec(local_mem) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in lmem.
<code>__declspec(local_mem) void *memcpy_lmem_lmem(__declspec(local_mem) void *dest, __declspec(local_mem) const void *src, unsigned int count);</code>	Copy from src in lmem to dest in lmem.
<code>__declspec(local_mem) void *memcpy_lmem_scratch(__declspec(local_mem) void *dest, __declspec(scratch) const void *src, unsigned int count);</code>	Copy from src in scratch to dest in lmem.
<code>__declspec(local_mem) void *memcpy_lmem_sram(__declspec(local_mem) void *dest, __declspec(sram) const void *src, unsigned int count);</code>	Copy from src in sram to dest in lmem.
<code>__declspec(scratch) void *memcpy_scratch_dram(__declspec(scratch) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in scratch.

Table 1. memcpy() Prototypes (Continued) (Sheet 2 of 2)

__declspec(scratch) void *memcpy_lmем(__declspec(scratch) void *dest, __declspec(local_mem) const void *src, unsigned int count);	Copy from src in lmem to dest in scratch.
__declspec(scratch) void *memcpy_scratch_scratch(__declspec(scratch) void *dest, __declspec(scratch) const void *src, unsigned int count);	Copy from src in scratch to dest in scratch.
__declspec(scratch) void *memcpy_scratch_sram(__declspec(scratch) void *dest, __declspec(sram) const void *src, unsigned int count);	Copy from src in sram to dest in scratch.
__declspec(sram) void *memcpy_sram_dram(__declspec(sram) void *dest, __declspec(dram) const void *src, unsigned int count);	Copy from src in dram to dest in sram.
__declspec(sram) void *memcpy_sram_lmем(__declspec(sram) void *dest, __declspec(local_mem) const void *src, unsigned int count);	Copy from src in local memory (lmem) to dest in sram.
__declspec(sram) void *memcpy_sram_scratch(__declspec(sram) void *dest, __declspec(scratch) const void *src, unsigned int count);	Copy from src in scratch to dest in sram.
__declspec(sram) void *memcpy_sram_sram(__declspec(sram) void *dest, __declspec(sram) const void *src, unsigned int count);	Copy from src in sram to dest in sram. This is identical to the memcpy() function.

1.5.2 memmove()

```
__declspec(sram) void *memmove(__declspec(sram) void *dest,
                               __declspec(sram) const void *src,
                               unsigned int count);
```

Like the standard C memcpy function, this function copies up to **count** characters from memory buffer **src** into buffer **dest** and returns **dest**. Unlike memcpy(), however, this function handles the case of overlapping buffers. Both memory buffers are in SRAM. This function correctly handles non optimal alignment cases.

Similar functions are provided for operation in all memory regions. [Table 2](#) summarizes the prototypes for each of the memory-specific memmove() functions.

Table 2. memmove() Prototypes (Sheet 1 of 2)

Prototype	Description
<code>__declspec(sram) void *memmove_sram_sram(__declspec(sram) void *dest, __declspec(sram) const void *src, unsigned int count);</code>	Copy from src in sram to dest in sram. This is identical to the memcpy() function.
<code>__declspec(sram) void *memmove_sram_scratch(__declspec(sram) void *dest, __declspec(scratch) const void *src, unsigned int count);</code>	Copy from src in scratch to dest in sram.
<code>__declspec(sram) void *memmove_sram_lmem(__declspec(sram) void *dest, __declspec(local_mem) const void *src, unsigned int count);</code>	Copy from src in local memory (lmem) to dest in sram.
<code>__declspec(sram) void *memmove_sram_dram(__declspec(sram) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in sram.
<code>__declspec(scratch) void *memmove_scratch_sram(__declspec(scratch) void *dest, __declspec(sram) const void *src, unsigned int count);</code>	Copy from src in sram to dest in scratch.
<code>__declspec(scratch) void *memmove_scratch_scratch(__declspec(scratch) void *dest, __declspec(scratch) const void *src, unsigned int count);</code>	Copy from src in scratch to dest in scratch.
<code>__declspec(scratch) void *memmove_scratch_lmem(__declspec(scratch) void *dest, __declspec(local_mem) const void *src, unsigned int count);</code>	Copy from src in lmem to dest in scratch.
<code>__declspec(scratch) void *memmove_scratch_dram(__declspec(scratch) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in scratch.
<code>__declspec(local_mem) void *memmove_lmem_sram(__declspec(local_mem) void *dest, __declspec(sram) const void *src, unsigned int count);</code>	Copy from src in sram to dest in lmem.
<code>__declspec(local_mem) void *memmove_lmem_scratch(__declspec(local_mem) void *dest, __declspec(scratch) const void *src, unsigned int count);</code>	Copy from src in scratch to dest in lmem.
<code>__declspec(local_mem) void *memmove_lmem_lmem(__declspec(local_mem) void *dest, __declspec(local_mem) const void *src, unsigned int count);</code>	Copy from src in lmem to dest in lmem.
<code>__declspec(local_mem) void *memmove_lmem_dram(__declspec(local_mem) void *dest, __declspec(dram) const void *src, unsigned int count);</code>	Copy from src in dram to dest in lmem.

Table 2. memmove() Prototypes (Continued) (Sheet 2 of 2)

<pre>__declspec(dram) void *memmove_dram_sram(__declspec(dram) void *dest, __declspec(sram) const void *src, unsigned int count);</pre>	<p>Copy from src in sram to dest in dram.</p>
<pre>__declspec(dram) void *memmove_dram_scratch(__declspec(dram) void *dest, __declspec(scratch) const void *src, unsigned int count);</pre>	<p>Copy from src in scratch to dest in dram.</p>
<pre>__declspec(dram) void *memmove_dram_lmem(__declspec(dram) void *dest, __declspec(local_mem) const void *src, unsigned int count);</pre>	<p>Copy from src in lmem to dest in dram.</p>
<pre>__declspec(dram) void *memmove_dram_dram(__declspec(dram) void *dest, __declspec(dram) const void *src, unsigned int count);</pre>	<p>Copy from src in dram to dest in dram.</p>

1.5.3 memset()

```

__declspec(sram) void *memset(__declspec(sram) void *dest,
                              CHAR c,
                              unsigned int count);
    
```

This function fills the aligned memory buffer **dest** in SRAM with repeated 8-bit characters **c** for the first **count** characters and returns **dest**.

Similar functions are provided for operation in all memory regions. [Table 3](#) summarizes the prototypes for each of the memory-specific memcmp() functions.

Table 3. memset() Prototypes

Prototype	Definition
<pre> __declspec(sram) void * memset_sram(__declspec(sram) void *dest, CHAR c, unsigned int count); </pre>	Fills the dest buffer in sram with count number of 8-bit characters specified by c . This is identical to the memset() function.
<pre> __declspec(scratch) void * memset_scratch(__declspec(scratch) void *dest, CHAR c, unsigned int count); </pre>	Fills the dest buffer in scratch with count number of 8-bit characters specified by c .
<pre> __declspec(local_mem) void * memset_lmem(__declspec(local_mem) void *dest, CHAR c, unsigned int count); </pre>	Fills the dest buffer in local memory with count number of 8-bit characters specified by c .
<pre> __declspec(dram) void * memset_dram(__declspec(dram) void *dest, CHAR c, unsigned int count); </pre>	Fills the dest buffer in dram with count number of 8-bit characters specified by c .

1.5.4 memcmp()

```
int memcmp(__declspec(sram) const void *buf1,
           __declspec(sram) const void *buf2,
           unsigned int count);
```

Compare the first **count** characters in two possibly misaligned memory buffers in SRAM. Perform unsigned comparison on each byte. Return a negative value if **buf1** is smaller than **buf2**, 0 if **buf1** is equal to **buf2**, or positive value if **buf1** is greater than **buf2**.

Similar functions are provided for operation in all memory regions. [Table 4](#) summarizes the prototypes for each of the memory-specific memcmp() functions.

Table 4. memcmp() Prototypes (Sheet 1 of 2)

Prototype	Description
<pre>int memcmp_sram_sram(__declspec(sram) const void *buf1, __declspec(sram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes in buf1 and buf2, both of which are in sram. This is identical to the memcmp() function.
<pre>int memcmp_sram_scratch(__declspec(sram) const void *buf1, __declspec(scratch) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in sram and buf2 in scratch.
<pre>int memcmp_sram_lmem(__declspec(sram) const void *buf1, __declspec(local_mem) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in sram and buf2 in lmem.
<pre>int memcmp_sram_dram(__declspec(sram) const void *buf1, __declspec(dram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in sram and buf2 in dram.
<pre>int memcmp_scratch_sram(__declspec(scratch) const void *buf1, __declspec(sram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in scratch and buf2 in sram.
<pre>int memcmp_scratch_scratch(__declspec(scratch) const void *buf1, __declspec(scratch) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in scratch and buf2 in scratch.
<pre>int memcmp_scratch_lmem(__declspec(scratch) const void *buf1, __declspec(local_mem) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in scratch and buf2 in lmem.
<pre>int memcmp_scratch_dram(__declspec(scratch) const void *buf1, __declspec(dram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in scratch and buf2 in dram.

Table 4. memcmp() Prototypes (Continued) (Sheet 2 of 2)

Prototype	Description
<pre>int memcmp_lmem_sram(__declspec(local_mem) const void *buf1, __declspec(sram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in local memory and buf2 in sram.
<pre>int memcmp_lmem_scratch(__declspec(local_mem) const void *buf1, __declspec(scratch) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in local memory and buf2 in scratch.
<pre>int memcmp_lmem_lmem(__declspec(local_mem) const void *buf1, __declspec(local_mem) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in local memory and buf2 in local memory.
<pre>int memcmp_lmem_dram(__declspec(local_mem) const void *buf1, __declspec(dram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in local memory and buf2 in dram.
<pre>int memcmp_dram_sram(__declspec(dram) const void *buf1, __declspec(sram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in dram and buf2 in sram
<pre>int memcmp_dram_scratch(__declspec(dram) const void *buf1, __declspec(scratch) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in dram and buf2 in scratch.
<pre>int memcmp_dram_lmem(__declspec(dram) const void *buf1, __declspec(local_mem) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in dram and buf2 in lmem.
<pre>int memcmp_dram_dram(__declspec(dram) const void *buf1, __declspec(dram) const void *buf2, unsigned int count);</pre>	Compares count number of bytes between buf1 in dram and buf2 in dram

1.5.5 memchr()

```
int memchr(__declspec(sram) const void *buf,
           CHAR c,
           unsigned int count);
```

Return offset to the first occurrence of 8-bit character **c** in memory buffer **buf** in SRAM, or -1 if none is found in the first **count** characters. Note that this function returns an offset rather than the address of the character as in standard C library.

Similar functions are provided for operation in all memory regions. [Table 5](#) summarizes the prototypes for each of the memory-specific memchr() functions.

Table 5. memchr() Prototypes

Prototype	Description
int memchr_sram(__declspec(sram) const void *buf, CHAR c, unsigned int count);	Returns an offset to the first occurrence of c in buf in sram.
int memchr_scratch(__declspec(scratch) const void *buf, CHAR c, unsigned int count);	Returns an offset to the first occurrence of c in buf in scratch.
int memchr_lmem(__declspec(local_mem) const void *buf, CHAR c, unsigned int count);	Returns an offset to the first occurrence of c in buf in lmem.
int memchr_dram(__declspec(dram) const void *buf, CHAR c, unsigned int count);	Returns an offset to the first occurrence of c in buf in dram.

1.6 string.h

The string.h file provides the same set of string manipulation functions as standard C. As with the functions in memory.h, string function names that do not specify a memory region work on strings in SRAM.

1.6.1 strlen()

```
unsigned int strlen(__declspec(sram) const CHAR *s);
```

Return length of string s in SRAM in bytes.

Similar functions are provided for operation in all memory regions. [Table 6](#) summarizes the prototypes for each of the memory-specific strlen() functions.

Table 6. strlen() Prototypes

Prototype	Description
unsigned int strlen_sram(__declspec(sram) const CHAR *s);	Returns the length of string s in sram.
unsigned int strlen_scratch(__declspec(scratch) const CHAR *s);	Returns the length of string s in scratch.
unsigned int strlen_lmem(__declspec(local_mem) const CHAR *s);	Returns the length of string s in lmem.
unsigned int strlen_dram(__declspec(dram) const CHAR *s);	Returns the length of string s in dram.

1.6.2 strcmp()

```
int strcmp(__declspec(sram) const CHAR *s1,
           __declspec(sram) const CHAR *s2);
```

Compare two strings in SRAM. Perform unsigned comparison on each CHAR.

Return:

negative value if: s1 is smaller than s2,
0 if s1 is equal to s2, and
positive value if s1 is greater than s2.

Similar functions are provided for operation in all memory regions. [Table 7](#) summarizes the prototypes for each of the memory-specific strcmp() functions.

Table 7. strcmp() Prototypes

Prototype	Description
unsigned int strcmp_sram(__declspec(sram) const CHAR *s1, __declspec(sram) const CHAR *s2);	Compares string s1 with string s2 in sram. This is identical to the strcmp() function.
unsigned int strcmp_scratch(__declspec(scratch) const CHAR *s1, __declspec(scratch) const CHAR *s2);	Compares string s1 with string s2 in scratch.
unsigned int strcmp_lmem(__declspec(local_mem) const CHAR *s1, __declspec(local_mem) const CHAR *s2);	Compares string s1 with string s2 in lmem.
unsigned int strcmp_dram(__declspec(dram) const CHAR *s1, __declspec(dram) const CHAR *s2);	Compares string s1 with string s2 in dram.

1.6.3 strncmp()

```
int strncmp(__declspec(sram) const CHAR *s1,
            __declspec(sram) const CHAR *s2,
            unsigned int count);
```

Compare two strings in SRAM up to a specified count of characters. Perform unsigned comparison on each CHAR. Cross product is not supported.

Return:

negative value if s1 is smaller than s2,
0 if s1 is equal to s2, or
positive value if s1 is greater than s2.

Similar functions are provided for operation in all memory regions. [Table 8](#) summarizes the prototypes for each of the memory-specific strncmp() functions.

Table 8. strncmp() Prototypes

Prototype	Description
int strncmp_sram(__declspec(sram) const CHAR *s1, __declspec(sram) const CHAR *s2, unsigned int count);	Compares count number of bytes in string s1 with string s2 in sram. This is identical to the strncmp() function.
int strncmp_scratch(__declspec(scratch) const CHAR *s1, __declspec(scratch) const CHAR *s2, unsigned int count);	Compares count number of bytes in string s1 with string s2 in scratch.
int strncmp_lmem(__declspec(local_mem) const CHAR *s1, __declspec(local_mem) const CHAR *s2, unsigned int count);	Compares count number of bytes in string s1 with string s2 in lmem.
int strncmp_dram(__declspec(dram) const CHAR *s1, __declspec(dram) const CHAR *s2, unsigned int count);	Compares count number of bytes in string s1 with string s2 in dram.

1.6.4 strcpy()

```
__declspec(sram) CHAR *strcpy(__declspec(sram) CHAR *dest,
                              __declspec(sram) const CHAR *src);
```

Copy source string **src** in SRAM to string **dest** in SRAM and return the original **dest**.

Similar functions are provided for operation in all memory regions. [Table 9](#) summarizes the prototypes for each of the memory-specific strcpy() functions.

Table 9. strcpy() Prototypes

Prototype	Description
__declspec(sram) CHAR *strcpy_sram(__declspec(sram) CHAR *dest, __declspec(sram) const CHAR *src);	Copies string src to string dest in sram. This is identical to the strcpy() function.
__declspec(scratch) CHAR *strcpy_scratch(__declspec(scratch) CHAR *dest, __declspec(scratch) const CHAR *src);	Copies string src to string dest in scratch.
__declspec(local_mem) CHAR *strcpy_lmem(__declspec(local_mem) CHAR *dest, __declspec(local_mem) const CHAR *src);	Copies string src to string dest in lmem.
__declspec(dram) CHAR *strcpy_dram(__declspec(dram) CHAR *dest, __declspec(dram) const CHAR *src);	Copies string src to string dest in dram.

1.6.5 strncpy()

```
__declspec(sram) CHAR *strncpy(__declspec(sram) CHAR *dest,
                               __declspec(sram) const CHAR *src,
                               unsigned int count);
```

Copy source string **src** in SRAM to string **dest** in SRAM, up to a specified **count** of characters, and return the original **dest**.

Similar functions are provided for operation in all memory regions. [Table 10](#) summarizes the prototypes for each of the memory-specific strncpy() functions.

Table 10. strncpy() Prototypes

Prototype	Description
<pre>__declspec(sram) CHAR *strncpy_sram(__declspec(sram) CHAR *dest, __declspec(sram) const CHAR *src, unsigned int count);</pre>	Copies count number of bytes from string src to string dest in sram. This is identical to the strncpy() function.
<pre>__declspec(scratch) CHAR *strncpy_scratch(__declspec(scratch) CHAR *dest, __declspec(scratch) const CHAR *src, unsigned int count);</pre>	Copies count number of bytes from string src to string dest in scratch.
<pre>__declspec(local_mem) CHAR *strncpy_lmem(__declspec(local_mem) CHAR *dest, __declspec(local_mem) const CHAR *src, unsigned int count);</pre>	Copies count number of bytes from string src to string dest in lmem.
<pre>__declspec(dram) CHAR *strncpy_dram(__declspec(dram) CHAR *dest, __declspec(dram) const CHAR *src, unsigned int count);</pre>	Copies count number of bytes from string src to string dest in dram.

1.6.6 strcat()

```
__declspec(sram) CHAR *strcat(__declspec(sram) CHAR *dest,
                             __declspec(sram) const CHAR *src);
```

Append string **src** in SRAM to string **dest** in SRAM and return the original **dest**.

Similar functions are provided for operation in all memory regions. [Table 11](#) summarizes the prototypes for each of the memory-specific strcat() functions.

Table 11. strcat() Prototypes

Prototype	Description
__declspec(sram) CHAR *strcat_sram(__declspec(sram) CHAR *dest, __declspec(sram) const CHAR *src);	Appends string src to string dest in sram. This is identical to the strcat() function.
__declspec(scratch) CHAR *strcat_scratch(__declspec(scratch) CHAR *dest, __declspec(scratch) const CHAR *src);	Appends string src to string dest in scratch.
__declspec(local_mem) CHAR *strcat_lmem(__declspec(local_mem) CHAR *dest, __declspec(local_mem) const CHAR *src);	Appends string src to string dest in lmem.
__declspec(dram) CHAR *strcat_dram(__declspec(dram) CHAR *dest, __declspec(dram) const CHAR *src);	Appends string src to string dest in dram.

1.6.7 strncat()

```
__declspec(sram) CHAR *strncat(__declspec(sram) CHAR *dest,
                              __declspec(sram) const CHAR *src,
                              unsigned int count);
```

Append string **src** in SRAM to string **dest** in SRAM, up to a specified **count** of characters, and return the original **dest**.

Similar functions are provided for operation in all memory regions. [Table 12](#) summarizes the prototypes for each of the memory-specific strncat() functions.

Table 12. strncat() Prototypes

Prototype	Description
<pre>__declspec(sram) CHAR *strncat_sram(__declspec(sram) CHAR *dest, __declspec(sram) const CHAR *src, unsigned int count);</pre>	<p>Appends count number of bytes from string src to string dest in sram. This is identical to the strncat() function.</p>
<pre>__declspec(scratch) CHAR *strncat_scratch(__declspec(scratch) CHAR *dest, __declspec(scratch) const CHAR *src, unsigned int count);</pre>	<p>Appends count number of bytes from string src to string dest in sram.</p>
<pre>__declspec(local_mem) CHAR *strncat_lmem(__declspec(local_mem) CHAR *dest, __declspec(local_mem) const CHAR *src, unsigned int count);</pre>	<p>Appends count number of bytes from string src to string dest in sram.</p>
<pre>__declspec(dram) CHAR *strncat_dram(__declspec(dram) CHAR *dest, __declspec(dram) const CHAR *src, unsigned int count);</pre>	<p>Appends count number of bytes from string src to string dest in sram.</p>

1.6.8 strchr()

```
__declspec(sram) CHAR *strchr(__declspec(sram)
                             const CHAR *s, CHAR c);
```

Return address of the first occurrence of character *c* (could be 0x00) in string *s* in SRAM, or NULL if none.

Similar functions are provided for operation in all memory regions. [Table 13](#) summarizes the prototypes for each of the memory-specific `strchr()` functions.

Table 13. `strchr()` Prototypes

Prototype	Description
<code>__declspec(sram) CHAR *strchr_sram(__declspec(sram) CHAR *s, CHAR *c);</code>	Returns the address of the first occurrence of <i>c</i> in string <i>s</i> in sram. This is identical to the <code>strchr()</code> function.
<code>__declspec(scratch) CHAR *strchr_scratch(__declspec(scratch) CHAR *s, CHAR *c);</code>	Returns the address of the first occurrence of <i>c</i> in string <i>s</i> in scratch.
<code>__declspec(local_mem) CHAR *strchr_lmem(__declspec(local_mem) CHAR *s, CHAR *c);</code>	Returns the address of the first occurrence of <i>c</i> in string <i>s</i> in lmem.
<code>__declspec(dram) CHAR *strchr_dram(__declspec(dram) CHAR *s, CHAR *c);</code>	Returns the address of the first occurrence of <i>c</i> in string <i>s</i> in dram.

1.6.9 strrchr()

```
__declspec(sram) CHAR *strrchr(__declspec(sram)
                             const CHAR *s, CHAR c);
```

Return address of the last occurrence of character *c* (could be 0x00) in string *s* in SRAM, or NULL if none.

Similar functions are provided for operation in all memory regions. [Table 14](#) summarizes the prototypes for each of the memory-specific `strrchr()` functions.

Table 14. strrchr() Prototypes

Prototype	Description
<code>__declspec(sram) CHAR *strrchr_sram(__declspec(sram) CHAR *s, CHAR *c);</code>	Returns the address of the last occurrence of <i>c</i> in string <i>s</i> in sram. This is identical to the <code>strchr()</code> function.
<code>__declspec(scratch) CHAR *strrchr_scratch(__declspec(scratch) CHAR *s, CHAR *c);</code>	Returns the address of the last occurrence of <i>c</i> in string <i>s</i> in scratch.
<code>__declspec(local_mem) CHAR *strrchr_lmem(__declspec(local_mem) CHAR *s, CHAR *c);</code>	Returns the address of the last occurrence of <i>c</i> in string <i>s</i> in lmem.
<code>__declspec(dram) CHAR *strrchr_dram(__declspec(dram) CHAR *s, CHAR *c);</code>	Returns the address of the last occurrence of <i>c</i> in string <i>s</i> in dram.

1.6.10 strstr()

```
__declspec(sram) CHAR *strstr(__declspec(sram) const CHAR *s1,
                             __declspec(sram) const CHAR *s2);
```

Return address of the first occurrence of sub-string **s2** in string **s1**, both in SRAM, or NULL if none.

Similar functions are provided for operation in all memory regions. [Table 15](#) summarizes the prototypes for each of the memory-specific strstr() functions.

Table 15. strstr() Prototypes

Prototype	Description
<pre>__declspec(sram) CHAR *strstr_sram(__declspec(sram) const CHAR *s1, __declspec(sram) const CHAR *s2);</pre>	Returns the address of the first occurrence of substring s2 in string s1 in sram. This is identical to the strstr() function.
<pre>__declspec(scratch) CHAR *strstr_scratch(__declspec(scratch) const CHAR *s1, __declspec(scratch) const CHAR *s2);</pre>	Returns the address of the first occurrence of substring s2 in string s1 in scratch.
<pre>__declspec(local_mem) CHAR *strstr_lmem(__declspec(local_mem) const CHAR *s1, __declspec(local_mem) const CHAR *s2);</pre>	Returns the address of the first occurrence of substring s2 in string s1 in lmem.
<pre>__declspec(dram) CHAR *strstr_dram(__declspec(dram) const CHAR *s1, __declspec(dram) const CHAR *s2);</pre>	Returns the address of the first occurrence of substring s2 in string s1 in dram.

1.6.11 strspn()

```
unsigned int strspn(__declspec(sram) const CHAR *s,
                  __declspec(sram) const CHAR *chset);
```

Return length of sub-string in string *s* in SRAM that consists entirely of characters in 8-bit character *chset* in SRAM.

Similar functions are provided for operation in all memory regions. [Table 16](#) summarizes the prototypes for each of the memory-specific `strspn()` functions.

Table 16. strspn() Prototypes

Prototype	Description
<pre>unsigned int strspn_sram(__declspec(sram) const CHAR *s, __declspec(sram) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters from chset in sram. This is identical to the <code>strspn()</code> function.
<pre>unsigned int strspn_scratch(__declspec(scratch) const CHAR *s, __declspec(scratch) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters from chset in scratch.
<pre>unsigned int strspn_lmem(__declspec(local_mem) const CHAR *s, __declspec(local_mem) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters from chset in lmem.
<pre>unsigned int strspn_dram(__declspec(dram) const CHAR *s, __declspec(dram) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters from chset in dram.

1.6.12 strcspn()

```
unsigned int strcspn(__declspec(sram) const CHAR *s,
                   __declspec(sram) const CHAR *chset);
```

Return length of sub-string in string *s* in SRAM that consists entirely of characters not in 8-bit character set *chset* in SRAM.

Similar functions are provided for operation in all memory regions. [Table 17](#) summarizes the prototypes for each of the memory-specific strcspn() functions.

Table 17. strcspn() Prototypes

Prototype	Description
<pre>unsigned int strcspn_sram(__declspec(sram) const CHAR *s, __declspec(sram) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters not contained in <i>chset</i> in sram. This is identical to the strcspn() function.
<pre>unsigned int strcspn_scratch(__declspec(scratch) const CHAR *s, __declspec(scratch) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters not contained in <i>chset</i> in scratch.
<pre>unsigned int strcspn_lmem(__declspec(local_mem) const CHAR *s, __declspec(local_mem) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters not contained in <i>chset</i> in lmem.
<pre>unsigned int strcspn_dram(__declspec(dram) const CHAR *s, __declspec(dram) const CHAR *chset);</pre>	Returns the length of the substring in string <i>s</i> that consists entirely of characters not contained in <i>chset</i> in dram.

1.6.13 strpbrk()

```
__declspec(sram) CHAR *strpbrk(__declspec(sram) const CHAR *s,  
                               __declspec(sram) const CHAR *chset);
```

Return address to the first character in string **s** that comes from 8-bit character set **chset**, both in SRAM, or NULL if none.

Similar functions are provided for operation in all memory regions. [Table 18](#) summarizes the prototypes for each of the memory-specific strpbrk() functions.

Table 18. strpbrk() Prototypes

Prototype	Description
<pre>__declspec(sram) CHAR *strpbrk_sram(__declspec(sram) const CHAR *s1, __declspec(sram) const CHAR *s2);</pre>	Returns the address of the first character in string s that comes from chset in sram. This is identical to the strpbrk() function.
<pre>__declspec(scratch) CHAR *strpbrk_scratch(__declspec(scratch) const CHAR *s1, __declspec(scratch) const CHAR *s2);</pre>	Returns the address of the first character in string s that comes from chset in scratch.
<pre>__declspec(local_mem) CHAR *strpbrk_lmem(__declspec(local_mem) const CHAR *s1, __declspec(local_mem) const CHAR *s2);</pre>	Returns the address of the first character in string s that comes from chset in lmem.
<pre>__declspec(dram) CHAR *strpbrk_dram(__declspec(dram) const CHAR *s1, __declspec(dram) const CHAR *s2);</pre>	Returns the address of the first character in string s that comes from chset in dram.

1.6.14 strtok()

```
__declspec(sram) CHAR *strtok(__declspec(sram) CHAR *s,  
                             __declspec(sram) const CHAR *delimit);
```

Return address to the next token in string *s* delimited by a character from the 8-bit character set *delimit*, both in SRAM,.

Similar functions are provided for operation in all memory regions. [Table 19](#) summarizes the prototypes for each of the memory-specific `strtok()` functions.

Table 19. strtok() Prototypes

Prototype	Description
<pre>__declspec(sram) CHAR *strtok_sram(__declspec(sram) CHAR *s, __declspec(sram) const CHAR *delimit);</pre>	Returns the address to the next token in string <i>s</i> delimited by a character from delimit in sram. This is identical to the <code>strtok()</code> function.
<pre>__declspec(scratch) CHAR *strtok_scratch(__declspec(scratch) CHAR *s, __declspec(scratch) const CHAR *delimit);</pre>	Returns the address to the next token in string <i>s</i> delimited by a character from delimit in scratch.
<pre>__declspec(local_mem) CHAR *strtok_lmem(__declspec(local_mem) CHAR *s, __declspec(local_mem) const CHAR *delimit);</pre>	Returns the address to the next token in string <i>s</i> delimited by a character from delimit in lmem.
<pre>__declspec(dram) CHAR *strtok_dram(__declspec(dram) CHAR *s, __declspec(dram) const CHAR *delimit);</pre>	Returns the address to the next token in string <i>s</i> delimited by a character from delimit in dram.

1.6.15 strtol()

```
long strtol(__declspec(sram) const CHAR *s,
            __declspec(sram) CHAR **ps_end,
            int base);
```

Convert string *s* in SRAM to a long integer using specified base (between 2 to 36, or 0 to interpret string in C-style). Return LONG_MAX or LONG_MIN if overflow or underflow, respectively. Adjust pointer **ps_end* to the first character that causes conversion to fail.

Similar functions are provided for operation in all memory regions. [Table 20](#) summarizes the prototypes for each of the memory-specific strtol() functions.

Table 20. strtol() Prototypes

Prototype	Description
<pre>long strtol_sram(__declspec(sram) const CHAR *s, __declspec(sram) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to a long integer using the specified base in sram. This is identical to the strtol() function.
<pre>long strtol_scratch(__declspec(scratch) const CHAR *s, __declspec(scratch) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to a long integer using the specified base in scratch.
<pre>long strtol_lmem(__declspec(local_mem) const CHAR *s, __declspec(local_mem) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to a long integer using the specified base in lmem.
<pre>long strtol_dram(__declspec(dram) const CHAR *s, __declspec(dram) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to a long integer using the specified base in dram.

1.6.16 strtoul()

```
unsigned long strtoul(__declspec(sram) const CHAR *s,
                    __declspec(sram) CHAR **ps_end,
                    int base);
```

Convert string *s* in SRAM to an unsigned long integer using specified base (between 2 to 36, or 0 to interpret string in C-style). Return `ULONG_MAX` if overflow. Adjust pointer **ps_end* to the first character that causes conversion to fail.

Similar functions are provided for operation in all memory regions. [Table 21](#) summarizes the prototypes for each of the memory-specific `strtoul()` functions.

Table 21. strtoul() Prototypes

Prototype	Description
<pre>long strtoul_sram(__declspec(sram) const CHAR *s, __declspec(sram) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to an unsigned long integer using the specified base in sram. This is identical to the <code>strtoul()</code> function.
<pre>long strtoul_scratch(__declspec(scratch) const CHAR *s, __declspec(scratch) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to an unsigned long integer using the specified base in scratch.
<pre>long strtoul_lmem(__declspec(local_mem) const CHAR *s, __declspec(local_mem) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to an unsigned long integer using the specified base in lmem.
<pre>long strtoul_dram(__declspec(dram) const CHAR *s, __declspec(dram) CHAR **ps_end, int base);</pre>	Converts string <i>s</i> to an unsigned long integer using the specified base in dram.

