

Libraries Guide



Arithmetic Libraries User Guide

Document #: 001-44477 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

© Cypress Semiconductor Corporation, 2006-2011. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™, Programmable System-on-Chip™ is trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

Contents

1. Introduction **5**

1.1 Arithmetic Libraries 5
1.1.1 Integer Functions 5
1.1.2 Floating-Point Functions 5
1.2 Using the Libraries with the ImageCraft Compiler 6
1.3 Using the Libraries with the HI-TECH Compiler 6
1.4 Revision History 6

2. Integer Arithmetic Functions **7**

2.1 Integer Division (8-bit) ... div_8x8_8 8
2.2 Integer Division (16-bit) ... div_16x16_16 9
2.3 Integer Division (32-bit) ... div_32x32_32 10
2.4 Unsigned Integer Division (8-bit) ... divu_8x8_8 12
2.5 Unsigned Integer Division (16-bit) ... divu_16x16_16 13
2.6 Unsigned Integer Division (32-bit) ... divu_32x32_32 14
2.7 Integer Remainder (8-bit) ... mod_8x8_8 16
2.8 Integer Remainder (16-bit) ... mod_16x16_16 17
2.9 Integer Remainder (32-bit) ... mod_32x32_32 18
2.10 Unsigned Integer Remainder (8-bit) ... modu_8x8_8 20
2.11 Unsigned Integer Remainder (16-bit) ... modu_16x16_16 21
2.12 Unsigned Integer Remainder (32-bit) ... modu_32x32_32 22
2.13 Integer Multiplication (8-bit) ... mul8 24
2.14 Integer Multiplication (16-bit) ... mul16 25
2.15 Integer Multiplication (32-bit) ... mul32 26
2.16 Integer Multiplication (8-bit) ... mul_8x8_16 28
2.17 Integer Multiplication (16-bit) ... mul_16x16_32 29
2.18 Unsigned Integer Multiplication (8-bit) ... mulu_8x8_16 31
2.19 Unsigned Integer Multiplication (16-bit) ... mulu_16x16_32 32

3. Floating-Point Functions **35**

3.1 Floating-Point Addition ... fpadd 36
3.2 Floating-Point Comparison ... fpcmp 38
3.3 Floating-Point Division ... fpdiv 39
3.4 Floating-Point Multiplication ... fpmul 41
3.5 Floating-Point Subtraction ... fpsub 43
3.6 Floating-Point Conversion ... fp2long 45
3.7 Floating-Point Conversion ... fp2ulong 46

3.8 Floating-Point Conversion long2fp 47
3.9 Floating-Point Conversion ulong2fp 48

1. Introduction

Cypress semiconductor released new arithmetic libraries with PSoC Designer™ 4.3. All functions in the libraries are available in both C and assembly language.

In this guide, you will find detailed descriptions about individual routines, as well as examples of how to call the routines from assembly language.

1.1 Arithmetic Libraries

For PSoC® parts having a MAC, the arithmetic libraries will automatically use it whenever suitable. Data for code sizes and cycle counts for the individual routines are provided. In general, the code is faster and takes up less space than the previous libraries. Internally, the arithmetic libraries reuse code whenever possible. Therefore, when using multiple functions, the combined code sizes and cycle counts will be larger than the actual numbers.

The compiler will automatically use the new libraries whenever possible. You need not do anything to accomplish this.

The libraries fit into two categories: integer functions and floating-point functions.

1.1.1 Integer Functions

The libraries for integer functions contain the normal functions for doing addition, subtraction, multiplication, and division. In addition, there are routines for calculating the modulus of two numbers. There are both signed and unsigned versions where needed. There are routines for “increased precision” calculations, example multiplying two 16-bit entities with a 32-bit result. This can be accomplished in C by first doing type casting, but the new routines are often more convenient and avoid passing multiple zero bytes on the stack to the normal library function. The C compiler is not aware of these special functions. You must call them explicitly to take advantage of these functions.

1.1.2 Floating-Point Functions

The implementation of floating-point arithmetic is completely rewritten. Except for one feature, the new version implements the IEEE-754 standard for floating-point numbers (single precision). The omission is that denormalized numbers are truncated to ‘0’. Managing denormalized numbers according to the standard increases the code size considerably. This compromise was created for the M8C microprocessor with its limited ROM space.

To make it more convenient to work with floating-point numbers in assembly, support for a new directive (DF – Define Floating-point Number) was added to the Assembler. See the *Assembly Language Guide* for more details.

1.2 Using the Libraries with the ImageCraft Compiler

The libraries described here are used by the ImageCraft compiler both explicitly when you call them and implicitly when the compiler replaces your code with function calls to these library functions. For example, the ImageCraft C Compiler will automatically replace the code shown with appropriate function calls to these libraries:

```
a = b * c;      /* in C */
d = f / Temp;   /* in C */
```

To use these libraries explicitly with the ImageCraft tools:

1. Include the .h or .inc file for the library described in this document. For example:
   ```
   #include <arith.h> /* in C */
   include “arith.inc“; in assembly
   ```
2. You can explicitly use any of the functions defined in the library in C or assembly.

1.3 Using the Libraries with the HI-TECH Compiler

By default, the HI-TECH C compiler uses different libraries instead of those documented here. See the *HI-TECH C© PRO for the PSoC Mixed-Signal Array* manual in the `docs` directory of your HI-TECH compiler installation. The HI-TECH compiler will use the alternate libraries both explicitly and implicitly in a manner similar to the way the ImageCraft C compiler uses these libraries.

You can use the libraries described here with the HI-TECH compiler explicitly only. It will not use the libraries in this document implicitly even if you include these libraries.

To use the libraries documented here explicitly with the HI-TECH compiler:

1. Include the .h or .inc file for the library you want to use. For example:
   ```
   #include <arith.h>   /* in C */
   include “arith.inc“; in assembly
   ```
2. Make sure that the path of the library is added to the compiler library search path.
3. You can explicitly use any of the functions defined in the library in C or assembly.

1.4 Revision History

Table 1-1. Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	May 29, 2008	FSU	This document was previously released in PSoC Designer but never had a document number. It was assigned a spec number and changes necessitated by template standards were added. Sections 1.2 and 1.3 were added. No other changes were made.
*A	June 22, 2011	RAVG/ PAUL	Document Title Changed. The word ‘Math’ and ‘Mathematical’ changed to ‘Arithmetic’.

2. Integer Arithmetic Functions

The function call mechanisms for the SMM (Small Memory Model) are presented in the “assembly” fragments. When these functions are used in the LMM (Large Memory Model), it is necessary to insert macros ("RAM_SETPAGE_CUR >'name of variable'") before access to each variable passed as input parameters or result parameter. These macros change the Current Page Pointer and guarantee that the variables (or virtual registers) used are on the page where direct memory instructions operate.

This chapter contains descriptions of the following integer arithmetic point functions:

- Integer Division (8-bit) div_8x8_8 on page 8.
- Integer Division (16-bit) div_16x16_16 on page 9.
- Integer Division (32-bit) div_32x32_32 on page 10.
- Unsigned Integer Division (8-bit) divu_8x8_8 on page 12.
- Unsigned Integer Division (16-bit) divu_16x16_16 on page 13.
- Unsigned Integer Division (32-bit) divu_32x32_32 on page 14.
- Integer Remainder (8-bit) mod_8x8_8 on page 16.
- Integer Remainder (16-bit) mod_16x16_16 on page 17.
- Integer Remainder (32-bit) mod_32x32_32 on page 18.
- Unsigned Integer Remainder (8-bit) modu_8x8_8 on page 20.
- Unsigned Integer Remainder (16-bit) modu_16x16_16 on page 21.
- Unsigned Integer Remainder (32-bit) modu_32x32_32 on page 22.
- Integer Multiplication (8-bit) mul8 on page 24.
- Integer Multiplication (16-bit) mul16 on page 25.
- Integer Multiplication (32-bit) mul32 on page 26.
- Integer Multiplication (8-bit) mul_8x8_16 on page 28.
- Integer Multiplication (16-bit) mul_16x16_32 on page 29.
- Unsigned Integer Multiplication (8-bit) mulu_8x8_16 on page 31.
- Unsigned Integer Multiplication (16-bit) mulu_16x16_32 on page 32.

2.1 Integer Division (8-bit) div_8x8_8

Description:

Divides first 8-bit signed parameter by second 8-bit signed parameter and returns the 8-bit signed result of division.

Synopsis:

```
#include <arith.h>
signed char div_8x8_8(signed char cOpr1, signed char cOpr2);
```

Assembly:

```
push      X             ; preserve X register if necessary
mov       X, [cOpr2]    ; push the second parameter cOpr2
mov       A, [cOpr1]    ; push the first parameter cOpr1
lcall     div_8x8_8     ; do the application
pop       X             ; restore the X register if necessary
mov       [r8s1], A     ; save result into ‘r8s1’ variable
```

Parameters:

`cOpr1(signed char):` first parameter in register A;

`cOpr2(signed char):` second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

signed char result of division `cOpr1`/`cOpr2` in register A.

Code Size:

Small Memory Model	Large Memory Model
71	85

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 475	Minimum: 515
Maximum: 575	Maximum: 615

2.2 Integer Division (16-bit) div_16x16_16

Description:

Divides first 16-bit signed parameter by second 16-bit signed parameter and returns the 16-bit signed result of division.

Synopsis:

```
#include <arith.h>
signed int div_16x16_16(signed int iOpr1, signed int iOpr2);
```

Assembly:

```
push      X                 ; preserve the X register if necessary
mov       A, [iOpr2+0]      ; push the second parameter iOpr2
push      A
mov       A, [iOpr2+1]
push      A
mov       A, [iOpr1+0]      ; push the first parameter wOpr1
push      A
mov       A, [iOpr1+1]
push      A
lcall     div_16x16_16      ; do the application
mov       [__rX], X         ; save LSB of result into [__rX]
add       SP, 252           ; pop the stack
pop       X                 ; restore the X register if necessary
mov       [r16s1+1], A      ; save the result into [r16s1] variable
mov       [r16s1+0], [__rX] ;
```

Parameters:

`iOpr1(signed int):` first parameter;

`iOpr2(signed int):` second parameter.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

16-bit signed result of division `iOpr1`/`iOpr2` in (MSB) X A (LSB).

Code Size:

Small Memory Model	Large Memory Model
128	150

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 1378	Minimum: 1435
Maximum: 1836	Maximum: 1893

2.3 Integer Division (32-bit) div_32x32_32

Description:

Divides first 32-bit signed parameter by second 32-bit signed parameter and stores the 32-bit signed result of division in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void div_32x32_32(signed long lOpr1, signed long lOpr2, signed long
*lRes);
```

Assembly:

```
push      X             ; preserve the X register if necessary
mov       A, >lRes      ; push the address of result variable
push      A
mov       A, <lRes
push      A
mov       A, [lOpr2+0]  ; push the second parameter lOpr2
push      A
mov       A, [lOpr2+1]
push      A
mov       A, [lOpr2+2]
push      A
mov       A, [lOpr2+3]
push      A
mov       A, [lOpr1+0]   ; push the first parameter lOpr1
push      A
mov       A, [lOpr1+1]
push      A
mov       A, [lOpr1+2]
push      A
mov       A, [lOpr1+3]
push      A
lcall     div_32x32_32   ; do the application
add       SP, 246        ; pop the stack
pop       X              ; restore the X register if necessary
```

Parameters:

`lOpr2(signed long):` first parameter;

`lOpr2(signed long):` second parameter;

`lRes(*signed long):` pointer to remainder of `lOpr1` and `lOpr2` parameters division.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The divisions result of the first signed long operand by second signed long operand is stored at the address specified by the third parameter.

2.3 Integer Division (32-bit) *(continued)* div_32x32_32

Code Size:

Small Memory Model	Large Memory Model
237	254

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 4397	Minimum: 4448
Maximum: 6149	Maximum: 6200

2.4 Unsigned Integer Division (8-bit) divu_8x8_8

Description:

Divides first 8-bit unsigned parameter by second 8-bit unsigned parameter and returns the 8-bit unsigned result of division.

Synopsis:

```
#include <arith.h>
unsigned char divu_8x8_8(unsigned char bOpr1, unsigned char bOpr2);
```

Assembly:

```
push      X          ; preserve the X register if necessary
mov       X, [bOpr2] ; push the second parameter bOpr2
mov       A, [bOpr1] ; push the first parameter bOpr1
lcall     divu_8x8_8 ; do the application
pop       X          ; restore the X register if necessary
mov       [r8u1], A  ; save the result into [r8u1] variable
```

Parameters:

`bOpr1(unsigned char):` first parameter in register A;

`bOpr2(unsigned char):` second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

unsigned char result of division `bOpr1`/`bOpr2` parameters in register A.

Code Size:

Small Memory Model	Large Memory Model
43	57

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 392	Minimum: 432
Maximum: 488	Maximum: 528

2.5 Unsigned Integer Division (16-bit) divu_16x16_16

Description:

Divides first 16-bit unsigned parameter by second 16-bit unsigned parameter and returns the 16-bit unsigned result of division.

Synopsis:

```
#include <arith.h>
unsigned int divu_16x16_16(unsigned int wOpr1, unsigned int wOpr2);
```

Assembly:

```
push    X                ; preserve the X register if necessary
mov     A, [wOpr2+0]     ; push the second parameter wOpr2
push    A
mov     A, [wOpr2+1]
push    A
mov     A, [wOpr1+0]     ; push the first parameter wOpr1
push    A
mov     A, [wOpr1+1]
push    A
lcall   divu_16x16_16    ; do the application
mov     [__rX],X         ; save LSB of result into [__rX]
add     SP, 252          ; pop the stack
pop     X                ; restore the X register if necessary
mov     [r16u1+1], A     ; save the result into [r16u1] variable
mov     [r16u1+0], [__rX] ;
```

Parameters:

`wOpr1(unsigned int):` first parameter;

`wOpr2(unsigned int):` second parameter.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

unsigned int result of division `wOpr1`/`wOpr2` parameters in (MSB)X A (LSB).

Code Size:

Small Memory Model	Large Memory Model
75	97

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 1287	Minimum: 1344
Maximum: 1703	Maximum: 1760

2.6 Unsigned Integer Division (32-bit) divu_32x32_32

Description:

Divides first 32-bit unsigned parameter by second 32-bit unsigned parameter and stores the 32-bit unsigned result of division in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void divu_32x32_32(unsigned long dOpr1, unsigned long dOpr2, unsigned
long *dRes);
```

Assembly:

```
push      X             ; preserve the X register if necessary
mov       A, >dRes      ; push the address of result variable
push      A
mov       A, <dRes
push      A
mov       A, [dOpr2+0]  ; push the second parameter dOpr2
push      A
mov       A, [dOpr2+1]
push      A
mov       A, [dOpr2+2]
push      A
mov       A, [dOpr2+3]
push      A
mov       A, [dOpr1+0]  ; push the first parameter dOpr1
push      A
mov       A, [dOpr1+1]
push      A
mov       A, [dOpr1+2]
push      A
mov       A, [dOpr1+3]
push      A
lcall     divu_32x32_32 ; do the application
add       SP, 246       ; pop the stack
pop       X             ; restore the X register if necessary
```

Parameters:

`dOpr1(unsigned long)` : first parameter;

`dOpr2(unsigned long)` : second parameter;

`dRes(unsigned long*)` : pointer to result of `dOpr1` and `dOpr2` parameters division.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The division’s result of the first unsigned long parameter by second unsigned long parameter is stored at the address specified by the third parameter.

2.6 Unsigned Integer Division (32-bit) *(continued)* divu_32x32_32

Code Size:

Small Memory Model	Large Memory Model
151	168

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 4320	Minimum: 4363
Maximum: 5984	Maximum: 6027

2.7 Integer Remainder (8-bit) mod_8x8_8

Description:

Divides first 8-bit signed parameter by second 8-bit signed parameter and returns the 8-bit signed char remainder of division.

Synopsis:

```
#include <arith.h>
signed char mod_8x8_8(signed char cOpr1, signed char cOpr2);
```

Assembly:

```
push      X             ; preserve the X register if necessary
mov       X, [cOpr2]    ; push the second parameter bOpr2
mov       A, [cOpr1]    ; push the first parameter bOpr1
lcall     mod_8x8_8     ; do the application
pop       X             ; restore the X register if necessary
mov       [r8s1], A     ; save the result into ‘r8s1’ variable
```

Parameters:

`cOpr1(signed char)` : first parameter in register A;

`cOpr2(signed char)` : second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

signed char remainder of division `cOpr1`/`cOpr2` parameters in register A.

Code Size:

Small Memory Model	Large Memory Model
70	84

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 474	Minimum: 514
Maximum: 574	Maximum: 614

2.8 Integer Remainder (16-bit) mod_16x16_16

Description:

Divides first 16-bit signed parameter by second 16-bit signed parameter and returns the 16-bit signed remainder of division.

Synopsis:

```
#include <arith.h>
signed int mod_16x16_16(signed int iOpr1, signed int iOpr2);
```

Assembly:

```
push      X                 ; preserve the X register if necessary
mov       A, [iOpr2+0]      ; push the second parameter iOpr2
push      A
mov       A, [iOpr2+1]
push      A
mov       A, [iOpr1+0]      ; push the first parameter iOpr1
push      A
mov       A, [iOpr1+1]
push      A
lcall     mod_16x16_16      ; do the application
mov       [__rX], X         ; save LSB of result into [__rX]
add       SP, 252           ; pop the stack
pop       X                 ; restore the X register if necessary
mov       [r16s1+1], A      ; save the result into [r16s1] variable
mov       [r16s1+0], [__rX] ;
```

Parameters:

`iOpr1(signed int)` : first parameter;

`iOpr2(signed int)` : second parameter.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

Signed int remainder of division `iOpr1`/`iOpr2` parameters in (MSB)X A (LSB).

Code Size:

Small Memory Model	Large Memory Model
126	148

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 1369	Minimum: 1426
Maximum: 1853	Maximum: 1910

2.9 Integer Remainder (32-bit) mod_32x32_32

Description:

Divides first 32-bit signed parameter by second 32-bit signed parameter and stores the 32-bit signed remainder of division in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void mod_32x32_32(signed long lOpr1, signed long lOpr2, signed long
*dRes);
```

Assembly:

```
push      X               ; preserve the X register if necessary
mov       A, >lRes        ; push the address of result variable
push      A
mov       A, <lRes
push      A
mov       A, [lOpr2+0]    ; push the second parameter lOpr2
push      A
mov       A, [lOpr2+1]
push      A
mov       A, [lOpr2+2]
push      A
mov       A, [lOpr2+3]
push      A
mov       A, [lOpr1+0]    ; push the first parameter lOpr1
push      A
mov       A, [lOpr1+1]
push      A
mov       A, [lOpr1+2]
push      A
mov       A, [lOpr1+3]
push      A
lcall     mod_32x32_32    ; do the application
add       SP, 246         ; pop the stack
pop       X               ; restore the X register if necessary
```

Parameters:

`lOpr1(signed long)` : first parameter;

`lOpr2(signed long)` : second parameter;

`lRes(signed long*)` : pointer to remainder of `lOpr1` and `lOpr2` parameters division.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The remainders division result of the first signed long parameter by second signed long parameter is stored at the address specified by the third parameter.

2.9 Integer Remainder (32-bit) *(continued)* mod_32x32_32

Code Size:

Small Memory Model	Large Memory Model
215	232

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 4325	Minimum: 4368
Maximum: 6143	Maximum: 6186

2.10 Unsigned Integer Remainder (8-bit) modu_8x8_8

Description:

Divides first 8-bit unsigned parameter by second 8-bit unsigned parameter and returns the 8-bit unsigned remainder of division.

Synopsis:

```
#include <arith.h>
unsigned char modu_8x8_8(unsigned char bOpr1, unsigned char bOpr2);
```

Assembly:

```
push      X             ; preserve the X register if necessary
mov       X, [bOpr2]    ; push the second parameter bOpr2
mov       A, [bOpr1]    ; push the first parameter bOpr1
lcall     modu_8x8_8    ; do the application
pop       X             ; restore the X register if necessary
mov       [r8u1], A     ; save the result into ‘r8u1’ variable
```

Parameters:

`bOpr1(unsigned char)` : first parameter in register A;

`bOpr2(unsigned char)` : second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

unsigned char remainder of division `bOpr1/Opr2` parameters in register A.

Code Size:

Small Memory Model	Large Memory Model
46	60

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 411	Minimum: 451
Maximum: 507	Maximum: 547

2.11 Unsigned Integer Remainder (16-bit) modu_16x16_16

Description:

Divides first 16-bit unsigned parameter by second 16-bit unsigned parameter and returns the 16-bit unsigned remainder of division.

Synopsis:

```
#include <arith.h>
unsigned int modu_16x16_16(unsigned int wOpr1, unsigned int wOpr2);
```

Assembly:

```
push      X                 ; preserve the X register if necessary
mov       A, [wOpr2+0]      ; push the second parameter wOpr2
push      A
mov       A, [wOpr2+1]
push      A
mov       A, [wOpr1+0]      ; push the first parameter wOpr1
push      A
mov       A, [wOpr1+1]
push      A
lcall     modu_16x16_16     ; do the application
mov       [__rX], X         ; save LSB of result into [__rX]
add       SP, 252           ; pop the stack
pop       X                 ; restore the X register if necessary
mov       [r16u1+1], A      ; save the result into [r16u1] variable
mov       [r16u1+0], [__rX] ;
```

Parameters:

`wOpr1(unsigned int)` : first parameter;

`wOpr2(unsigned int)` : second parameter.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

unsigned int remainder of division `wOpr1`/`wOpr2` parameters in (MSB)X A(LSB).

Code Size:

Small Memory Model	Large Memory Model
75	97

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 1285	Minimum: 1342
Maximum: 1701	Maximum: 1758

2.12 Unsigned Integer Remainder (32-bit) modu_32x32_32

Description:

Divides first 32-bit unsigned parameter by second 32-bit unsigned parameter and stores the 32-bit unsigned remainder of division in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void modu_32x32_32(unsigned long dOpr1, unsigned long dOpr2, unsigned
long *dRes);
```

Assembly:

```
push      X               ; preserve the X register if necessary
mov       A, >dRes        ; push the address of dRes
push      A
mov       A, <dRes
push      A
mov       A, [dOpr2+0]    ; push the second parameter dOpr2
push      A
mov       A, [dOpr2+1]
push      A
mov       A, [dOpr2+2]
push      A
mov       A, [dOpr2+3]
push      A
mov       A, [dOpr1+0]    ; push the first parameter dOpr1
push      A
mov       A, [dOpr1+1]
push      A
mov       A, [dOpr1+2]
push      A
mov       A, [dOpr1+3]
push      A
lcall     modu_32x32_32   ; do the application
add       SP, 246         ; pop the stack
pop       X               ; restore the X register if necessary
```

Parameters:

`dOpr1(unsigned long)` : first parameter;

`dOpr2(unsigned long)` : second parameter;

`dRes(unsigned long*)` : pointer to remainder of `dOpr1` and `dOpr2` parameters division.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The remainders result of the first unsigned long operand by second unsigned long operand is stored at the address specified by the third parameter.

2.12 Unsigned Integer Remainder (32-bit) *(continued)* modu_32x32_32

Code Size:

Small Memory Model	Large Memory Model
131	148

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 4255	Minimum: 4298
Maximum: 5919	Maximum: 5962

2.13 Integer Multiplication (8-bit) mul8

Description:

Multiplies two 8-bit parameters and returns the low 8-bit multiplication result.

Synopsis:

```
#include <arith.h>
unsigned char mul8(unsigned char cOpr1,unsigned char cOpr2);
```

Assembly:

```
push    X           ; preserve the X register if necessary
mov     X, [cOpr2]  ; push the second parameter cOpr2
mov     A, [cOpr1]  ; push the first parameter cOpr1
lcall   mul8        ; do the application
pop     X           ; restore the X register if necessary
mov     [r8u1], A   ; save the result into ‘r8u1’ variable
```

Parameters:

`cOpr1:` first unsigned char parameter in register A;

`cOpr2:` second unsigned char parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

Low 8-bit of multiplication result in register A.

Code Size:

	Small Memory Model	Large Memory Model
MAC	19	30
No MAC	28	42

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 66	Minimum: 98
	Maximum: 66	Maximum: 98
No MAC	Minimum: 84	Minimum: 124
	Maximum: 300	Maximum: 340

2.14 Integer Multiplication (16-bit) mul16

Description:

Multiplies two 16-bit parameters and returns the low 16-bit multiplication result.

Synopsis:

```
#include <arith.h>
unsigned int mul16(unsigned int wOpr1, unsigned int wOpr2);
```

Assembly:

```
push    X               ; preserve the X register if necessary
mov     A, [wOpr2+0]    ; push the second parameter iOpr2
push    A
mov     A, [wOpr2+1]
push    A
mov     A, [wOpr1+0]    ; push the first parameter wOpr1
push    A
mov     A, [wOpr1+1]
push    A
lcall   mul16           ; do the application
mov     [__rX],X        ; save LSB of result into [__rX]
add     SP, 252         ; pop the stack
pop     X               ; restore the X register if necessary
mov     [r16u1+1], A    ; save the result into [r16u1] variable
mov     [r16u1+0], [__rX] ;
```

Parameters:

`wOpr1(unsigned int)` : first parameter;
`wOpr2(unsigned int)` : second parameter.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

Low 16-bit result of multiplication `wOpr1`*`wOpr2` parameters in (MSB)X A(LSB).

Code Size:

	Small Memory Model	Large Memory Model
MAC	124	149
No MAC	71	93

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 626	Minimum: 683
	Maximum: 674	Maximum: 731
No MAC	Minimum: 1048	Minimum: 1105
	Maximum: 1464	Maximum: 1521

2.15 Integer Multiplication (32-bit) mul32

Description:

Multiplies two 32-bit parameters and stores the low 32-bit multiplication result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void mul32(unsigned long dOpr1, unsigned long dOpr1, unsigned long
*dRes);
```

Assembly:

```
push    X             ; preserve the X register if necessary
mov     A, >dRes      ; push the address of result variable
push    A
mov     A, <dRes
push    A
mov     A, [dOpr2+0]  ; push the second parameter dOpr2
push    A
mov     A, [dOpr2+1]
push    A
mov     A, [dOpr2+2]
push    A
mov     A, [dOpr2+3]
push    A
mov     A, [dOpr1+0]   ; push first parameter dOpr1
push    A
mov     A, [dOpr1+1]
push    A
mov     A, [dOpr1+2]
push    A
mov     A, [dOpr1+3]
push    A
lcall   mul32          ; do the application
add     SP, 246        ; pop the stack
pop     X              ; restore the X register if necessary
```

Parameters:

`dOpr1(unsigned long)` : first parameter;

`dOpr2(unsigned long)` : second operand in register X;

`dRes(unsigned long*)` : pointer to result of `dOpr1` and `dOpr2` parameters multiplication.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first two parameters multiplication is stored at the address specified by the third parameter.

2.15 Integer Multiplication (32-bit) *(continued)* mul32

Code Size:

	Small Memory Model	Large Memory Model
MAC	203	223
No MAC	108	125

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 1034	Minimum: 1077
	Maximum: 1106	Maximum: 1149
No MAC	Minimum: 2819	Minimum: 2862
	Maximum: 4483	Maximum: 4526

2.16 Integer Multiplication (8-bit) mul_8x8_16

Description:

Multiplies two 8-bit signed operands and returns the 16-bit signed multiplication result.

Synopsis:

```
#include <arith.h>
signed int mul_8x8_16(signed char cOpr1, signed char cOpr2);
```

Assembly:

```
push      X                 ; preserve the X register if necessary
mov       X, [cOpr2]        ; push the second parameter cOpr2
mov       A, [cOpr1]        ; push the first parameter cOpr1
lcall     mul_8x8_16        ; do the application
mov       [__rX], X         ; save LSB of result into [__rX]
pop       X                 ; restore the X register if necessary
mov       [r16s1+1], A      ; save the result into [r16s1] variable
mov       [r16s1+0], [__rX] ;
```

Parameters:

`cOpr1(signed char)` : first parameter in register A;

`cOpr2(signed char)` : second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

signed int multiplication result in register (MSB)X A(LSB).

Code Size:

	Small Memory Model	Large Memory Model
MAC	27	44
No MAC	61	78

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 91	Minimum: 148
	Maximum: 91	Maximum: 148
No MAC	Minimum: 406	Minimum: 463
	Maximum: 466	Maximum: 523

2.17 Integer Multiplication (16-bit) mul_16x16_32

Description:

Multiplies two 16-bit signed parameters and stores the 32-bit signed multiplication result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void   mul_16x16_32(signed int wOpr1, signed int wOpr2, signed long
*dRes);
```

Assembly:

```
push      X               ; preserve the X register if necessary
mov       A, >lRes        ; push the address of result variable
push      A
mov       A, <lRes
push      A
mov       A, [iOpr2+0]    ; push the second parameter iOpr2
push      A
mov       A, [iOpr2+1]
push      A
mov       A, [iOpr1+0]    ; push the first parameter iOpr1
push      A
mov       A, [iOpr1+1]
push      A
lcall     mul_16x16_32    ; do the application
add       SP, 250         ; pop the stack
pop       X               ; restore the X register if necessary
```

Parameters:

`iOpr1(signed int)` : first parameter;

`iOpr2(signed int)` : second parameter;

`lRes(signed long*)` : pointer to result of `iOpr1` and `iOpr2` parameters multiplication.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first two parameters multiplication is stored at the address specified by the third parameter.

Code Size:

	Small Memory Model	Large Memory Model
MAC	165	182
No MAC	119	133

2.17 Integer Multiplication (16-bit) *(continued)* mul_16x16_32

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 713	Minimum: 756
	Maximum: 813	Maximum: 856
No MAC	Minimum: 1161	Minimum: 1204
	Maximum: 1622	Maximum: 1665

2.18 Unsigned Integer Multiplication (8-bit) mulu_8x8_16

Description:

Multiplies two 8-bit unsigned parameters and returns the 16-bit unsigned multiplication result.

Synopsis:

```
#include <arith.h>
unsigned char mulu_8x8_16(unsigned char bOpr1, unsigned char bOpr2);
```

Assembly:

```
push      X                 ; preserve the X register if necessary
mov       X, [bOpr2]        ; push the second parameter bOpr2
mov       A, [bOpr1]        ; push the first parameter bOpr1
lcall     mulu_8x8_16       ; do the application
mov       [__rX], X         ; save LSB of result into [__rX]
pop       X                 ; restore the X register if necessary
mov       [r16u1+1], A      ; save the result into [r16s1] variable
mov       [r16u1+0], [__rX] ;
```

Parameters:

`bOpr1(unsigned char)` : first parameter in register A;

`bOpr2(unsigned char)` : second parameter in register X.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

Unsigned int multiplication result returns in registers (MSB)X A(LSB).

Code Size:

	Small Memory Model	Large Memory Model
MAC	44	64
No MAC	45	65

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 126	Minimum: 75
	Maximum: 138	Maximum: 187
No MAC	Minimum: 375	Minimum: 432
	Maximum: 423	Maximum: 480

2.19 Unsigned Integer Multiplication (16-bit) mulu_16x16_32

Description:

Multiplies two 16-bit unsigned parameters and stores the 32-bit unsigned multiplication result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void mulu_16x16_32(unsigned int wOpr1, unsigned int wOpr2, unsigned
long *dRes);
```

Assembly:

```
push      X               ; preserve the X register if necessary
mov       A, >dRes        ; push the address of result variable
push      A
mov       A, <dRes
push      A
mov       A, [wOpr2+0]    ; push the second parameter wOpr2
push      A
mov       A, [wOpr2+1]
push      A
mov       A, [wOpr1+0]    ; push the first parameter wOpr1
push      A
mov       A, [wOpr1+1]
push      A
lcall     mulu_16x16_32   ; do the converting
add       SP, 250         ; pop the stack
pop       X               ; restore the X register if necessary
```

Parameters:

`wOpr1(unsigned int)` : first parameter;

`wOpr2(unsigned int)` : second parameter;

`dRes(unsigned long*)` : pointer to result of `wOpr1` and `wOpr2` multiplication.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first two parameters multiplication is stored at the address specified by the third parameter.

Code Size:

	Small Memory Model	Large Memory Model
MAC	140	160
No MAC	87	103

2.19 Unsigned Integer Multiplication (16-bit) *(continued)* mulu_16x16_32

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 686	Minimum: 729
	Maximum: 734	Maximum: 777
No MAC	Minimum: 1108	Minimum: 1149
	Maximum: 1524	Maximum: 1565

3. Floating-Point Functions

The function call mechanisms for the SMM (Small Memory Model) are presented in the “assembly” fragments. When these functions are used in the LMM (Large Memory Model), it is necessary to insert macros ("RAM_SETPAGE_CUR >'name of variable"') before access to each variable passed as input parameters or result parameter. These macros change the Current Page Pointer and guarantee that the variables (or virtual registers) used are on the page where the direct memory instructions operate.

This chapter encompasses the following floating-point functions:

- Floating-Point Addition fpadd on page 36.
- Floating-Point Comparison fpcmp on page 38.
- Floating-Point Division fpdiv on page 39.
- Floating-Point Multiplication fpmul on page 41.
- Floating-Point Subtraction fpsub on page 43.
- Floating-Point Conversion fp2long on page 45.
- Floating-Point Conversion fp2ulong on page 46.
- Floating-Point Conversion long2fp on page 47.
- Floating-Point Conversion ulong2fp on page 48.

3.1 Floating-Point Addition fpadd

Description:

Adds two floating-point numbers and stores the result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void fpadd(float fOpr1, float fOpr2, float *fRes);
```

Assembly:

```
push    X               ; preserve X register if necessary
mov     A, >fRes
push    A               ; push the address of result variable
mov     A, <fRes
push    A
mov     A, [fOpr2+0]    ; push the second parameter fOpr2
push    A
mov     A, [fOpr2+1]
push    A
mov     A, [fOpr2+2]
push    A
mov     A, [fOpr2+3]
push    A
mov     A, [fOpr1+0]    ; push the first parameter fOpr1
push    A
mov     A, [fOpr1+1]
push    A
mov     A, [fOpr1+2]
push    A
mov     A, [fOpr1+3]
push    A
lcall   fpadd            ; do the addition
add     SP, 246          ; pop the stack
pop     X                ; restore the X register if necessary
```

Parameters:

`fOpr1 (float)` : first floating-point parameter;

`fOpr2 (float)` : second floating-point parameter;

`fRes (float*)` : pointer to the result of the sum of `fOpr1` and `fOpr2` float-point parameters.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The sum of the first two parameters is stored at the address specified by the third parameter.

Code Size:

Small Memory Model	Large Memory Model
461	478

3.1 Floating-Point Addition *(continued)* fpadd

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 484	Minimum: 527
Maximum: 2150	Maximum: 2193

3.2 Floating-Point Comparison fpcmp

Description:

Compares two floating-point numbers, A and B. Returns 0 if A == B, 1 if A >B, or -1 if A < B.

Synopsis:

```
#include <arith.h>
signed char fpcmp(float fOpr1, float fOpr2);
```

Assembly:

```
push    X           ; preserve X register if necessary
mov     A, [fOpr2+0] ; push the second parameter fOpr2
push    A
mov     A, [fOpr2+1]
push    A
mov     A, [fOpr2+2]
push    A
mov     A, [fOpr2+3]
push    A
mov     A, [fOpr1+0] ; push the first parameter fOpr1
push    A
mov     A, [fOpr1+1]
push    A
mov     A, [fOpr1+2]
push    A
mov     A, [fOpr1+3]
push    A
lcall   fpcmp       ; do the comparison
add     SP,  248    ; pop the stack
pop     X           ; restore the X register if necessary
mov     [r8s1], A   ; save result into r8s1 variable
```

Parameters:

`fOpr1 (float)` : first floating-point parameter;

`fOpr2 (float)` : second floating-point parameter;

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP page pointer register is modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

Signed char result from comparing `fOpr1` and `fOpr2` in reg. A

Code Size:

Small Memory Model	Large Memory Model
109	125

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 160	Minimum: 200
Maximum: 342	Maximum: 382

3.3 Floating-Point Division fpdiv

Description:

Divides first floating-point number by second floating-point number and stores the result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void fpdiv(float fOpr1, float fOpr2, float *fRes);
```

Assembly:

```
push    X           ; preserve X register if necessary
mov     A, >fRes    ; push the address of result variable
push    A
mov     A, <fRes
push    A
mov     A, [fOpr2+0] ; push the second parameter fOpr2
push    A
mov     A, [fOpr2+1]
push    A
mov     A, [fOpr2+2]
push    A
mov     A, [fOpr2+3]
push    A
mov     A, [fOpr1+0]  ; push the first parameter fOpr1
push    A
mov     A, [fOpr1+1]
push    A
mov     A, [fOpr1+2]
push    A
mov     A, [fOpr1+3]
push    A
lcall   fpdiv         ; do the division
add     SP, 246       ; pop the stack
pop     X             ; restore the X register if necessary
```

Parameters:

`fOpr1 (float)` : first floating-point parameter;

`fOpr2 (float)` : second floating-point parameter;

`fRes (float*)` : pointer to result of `fOpr1` divided by `fOpr2`.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The division of the first floating-point parameter by second is stored at the address specified by the third parameter.

3.3 Floating-Point Division *(continued)* fpdiv

Code Size:

Small Memory Model	Large Memory Model
432	449

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 487	Minimum: 530
Maximum: 4280	Maximum: 4323

3.4 Floating-Point Multiplication fpmul

Description:

Multiplies two floating-point numbers and stores the result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void fpmul(float fOpr1, float fOpr2, float *fRes);
```

Assembly:

```
push      X            ; preserve X register if necessary
mov       A, >fRes     ; push the address of result variable
push      A
mov       A, <fRes
push      A
mov       A, [fOpr2+0] ; push the second parameter fOpr2
push      A
mov       A, [fOpr2+1]
push      A
mov       A, [fOpr2+2]
push      A
mov       A, [fOpr2+3]
push      A
mov       A, [fOpr1+0] ; push the first parameter fOpr1
push      A
mov       A, [fOpr1+1]
push      A
mov       A, [fOpr1+2]
push      A
mov       A, [fOpr1+3]
push      A
lcall     fpmul       ; do the multiplication
add       SP, 246     ; pop the stack
pop       X           ; restore the X register if necessary
```

Parameters:

`fOpr1 (float):` first floating-point parameter;

`fOpr2 (float):` second floating-point parameter;

`fRes (float*):` pointer to result of `fOpr1` and `fOpr2` multiplication.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first two parameters multiplication is stored at the address specified by the third parameter.

3.4 Floating-Point Multiplication *(continued)* fpmul

Code Size:

	Small Memory Model	Large Memory Model
MAC	538	558
No MAC	406	423

Cycle Count:

	Small Memory Model	Large Memory Model
MAC	Minimum: 463	Minimum: 506
	Maximum: 2047	Maximum: 2090
No MAC	Minimum: 463	Minimum: 506
	Maximum: 3413	Maximum: 3456

3.5 Floating-Point Subtraction fpsub

Description:

Subtracts second floating-point numbers from first floating-point numbers and stores the result in specified memory location. The address of either of the two parameters can be used for the result.

Synopsis:

```
#include <arith.h>
void fpsub(float fOpr1, float fOpr2, float *fRes);
```

Assembly:

```
push      X             ; preserve X register if necessary
mov       A, >fRes      ; push the address of result variable
push      A
mov       A, <fRes
push      A
mov       A, [fOpr2+0]  ; push the second parameter fOpr2
push      A
mov       A, [fOpr2+1]
push      A
mov       A, [fOpr2+2]
push      A
mov       A, [fOpr2+3]
push      A
mov       A, [fOpr1+0]  ; push the first parameter fOpr1
push      A
mov       A, [fOpr1+1]
push      A
mov       A, [fOpr1+2]
push      A
mov       A, [fOpr1+3]
push      A
lcall     fpsub         ; do the subtraction
add       SP, 246       ; pop the stack
pop       X             ; restore the X register if necessary
```

Parameters:

`fOpr1 (float)` : first floating-point parameter;

`fOpr2 (float)` : second floating-point parameter;

`fRes (float*)` : pointer to result of `fOpr1` minus `fOpr2`.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first two parameters subtraction is stored at the address specified by the third parameter.

3.5 Floating-Point Subtraction *(continued)* fpsub

Code Size:

Small Memory Model	Large Memory Model
468	485

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 505	Minimum: 548
Maximum: 2171	Maximum: 2214

3.6 Floating-Point Conversion fp2long

Description:

Converts a floating-point number to a 32-bit signed integer.

Synopsis:

```
#include <arith.h>
void fp2long(float fOpr, signed long *lRes);
```

Assembly:

```
push      X            ; preserve X register if necessary
mov       A, >lRes     ; push the address of result variable
push      A
mov       A, <lRes
push      A
mov       A, [fOpr+0]  ; push the first parameter fOpr
push      A
mov       A, [fOpr+1]
push      A
mov       A, [fOpr+2]
push      A
mov       A, [fOpr+3]
push      A
lcall     fp2long      ; do the conversionconversion
add       SP, 250      ; pop the stack
pop       X            ; restore the X register if necessary
```

Parameters:

`fOpr (float)` : first floating-point parameter;

`lRes (long*)` : pointer to signed long result from conversion fOpr.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first parameter conversion is stored at the address specified by the second parameter.

Code Size:

Small Memory Model	Large Memory Model
144	158

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 262	Minimum: 297
Maximum: 1642	Maximum: 1677

3.7 Floating-Point Conversion fp2ulong

Description:

Converts a floating-point number to a 32-bit unsigned integer.

Synopsis:

```
#include <arith.h>
void fp2ulong(float fOpr, unsigned long *dRes);
```

Assembly:

```
push      X            ; preserve X register if necessary
mov       A, >dRes     ; push the address of result variable
push      A
mov       A, <dRes
push      A
mov       A, [fOpr+0]  ; push the parameter fOpr
push      A
mov       A, [fOpr+1]
push      A
mov       A, [fOpr+2]
push      A
mov       A, [fOpr+3]
push      A
lcall     fp2ulong     ; do the conversion
add       SP, 250      ; pop the stack
pop       X            ; restore the X register if necessary
```

Parameters:

`fOpr (float)` : first floating-point parameter;

`dRes (unsigned long*)` : pointer to unsigned long result from converting `fOpr`.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first parameter conversion is stored at the address specified by the second parameter.

Code Size:

Small Memory Model	Large Memory Model
141	155

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 255	Minimum: 290
Maximum: 1635	Maximum: 1670

3.8 Floating-Point Conversion long2fp

Description:

Converts a 32-bit signed integer to a floating-point number.

Synopsis:

```
#include <arith.h>
void long2fp(signed long lOpr, float* fRes);
```

Assembly:

```
push      X            ; preserve X register if necessary
mov       A, >fRes     ; push the address of result variable
push      A
mov       A, <fRes
push      A
mov       A, [lOpr+0]  ; push the parameter lOpr
push      A
mov       A, [lOpr+1]
push      A
mov       A, [lOpr+2]
push      A
mov       A, [lOpr+3]
push      A
lcall     long2fp      ; do the conversion
add       SP, 250      ; pop the stack
pop       X            ; restore the X register if necessary
```

Parameters:

`lOpr (long) :` first signed long parameter;

`fRes (float*) :` pointer to floating-point result from converting `lOpr`.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first parameter conversion is stored at the address specified by the second parameter.

Code Size:

Small Memory Model	Large Memory Model
170	184

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 254	Minimum: 289
Maximum: 1551	Maximum: 1586

3.9 Floating-Point Conversion ulong2fp

Description:

Converts a 32-bit unsigned integer to a floating-point number.

Synopsis:

```
#include <arith.h>
void ulong2fp(unsigned long dOpr, float* fRes);
```

Assembly:

```
push    X             ; preserve X register if necessary
mov     A, >fRes      ; push the address of result variable
push    A
mov     A, <fRes
push    A
mov     A, [dOpr+0]   ; push the parameter dOpr
push    A
mov     A, [dOpr+1]
push    A
mov     A, [dOpr+2]
push    A
mov     A, [dOpr2+3]
push    A
lcall   ulong2fp      ; do the conversion
add     SP, 250       ; pop the stack
pop     X             ; restore the X register if necessary
```

Parameters:

`dOpr (unsigned long) :` first unsigned long operand;

`fRes (float*) :` pointer to floating-point number result from converting `dOpr`.

Side Effects:

The A and X registers are modified by this function implementation. Currently, only the CUR_PP and MVW_PP page pointers registers are modified. When necessary, it is the calling function's responsibility to preserve the values across calls to this function.

Return Value:

None. The result of the first parameter conversion is stored at the address specified by the second parameter.

Code Size:

Small Memory Model	Large Memory Model
134	148

Cycle Count:

Small Memory Model	Large Memory Model
Minimum: 227	Minimum: 262
Maximum: 1429	Maximum: 1464