Writing CSUBs for MMBasic on the PicoMite

*A Comprehensive Guide to Embedding C Code in MMBasic Programs*

For Raspberry Pi Pico (RP2040) and Pico 2 (RP2350)

**Table of Contents**

[Introduction 2](#__RefHeading___Toc390_400029848)

[What is a CSUB? 2](#__RefHeading___Toc392_400029848)

[Getting Started 2](#__RefHeading___Toc394_400029848)

[Required Tools 2](#__RefHeading___Toc396_400029848)

[C Function Structure 3](#__RefHeading___Toc398_400029848)

[Basic Template 3](#__RefHeading___Toc400_400029848)

[Key Components 3](#__RefHeading___Toc402_400029848)

[Complete Working Example 3](#__RefHeading___Toc404_400029848)

[Integer Power Function 3](#__RefHeading___Toc406_400029848)

[Using in BASIC 4](#__RefHeading___Toc408_400029848)

[Compilation Process 5](#__RefHeading___Toc410_400029848)

[Compiling for RP2040 (Cortex-M0+) 5](#__RefHeading___Toc412_400029848)

[Compiling for RP2350 (Cortex-M33) 5](#__RefHeading___Toc414_400029848)

[Converting to CSUB Format 5](#__RefHeading___Toc416_400029848)

[Parameter Handling 5](#__RefHeading___Toc418_400029848)

[Parameter Types 5](#__RefHeading___Toc420_400029848)

[Accessing Parameters in C 5](#__RefHeading___Toc422_400029848)

[Best Practices 6](#__RefHeading___Toc424_400029848)

[Advanced Topics 7](#__RefHeading___Toc426_400029848)

[Using the Helper Toolkit 7](#__RefHeading___Toc428_400029848)

[Saving to Library 7](#__RefHeading___Toc430_400029848)

[Accessing MMBasic Internals 7](#__RefHeading___Toc432_400029848)

[Troubleshooting 8](#__RefHeading___Toc434_400029848)

[Performance Comparison 8](#__RefHeading___Toc436_400029848)

[Additional Resources 8](#__RefHeading___Toc438_400029848)

[Conclusion 9](#__RefHeading___Toc440_400029848)

# Introduction

CSUBs (C Subroutines) are a powerful feature of MMBasic that allows you to embed compiled ARM machine code directly into your BASIC programs. This provides native processor execution speed for performance-critical sections of code while maintaining the ease and flexibility of BASIC for the rest of your program.

**Key Benefits:**

* 7-10x performance improvement over interpreted BASIC
* No firmware rebuild required
* Drop-in replacement for slow BASIC loops
* Native ARM Cortex-M0+ (RP2040) or Cortex-M33 (RP2350) execution
* Can be saved to library for reuse across programs

For example, a Mandelbrot set renderer can be accelerated from 70 seconds to just 10 seconds using a CSUB implementation.

# What is a CSUB?

A CSUB contains ARM processor machine code instructions stored in hexadecimal format within your BASIC program. When your program runs, MMBasic automatically loads this hexadecimal code into memory as binary and you can call it just like a normal BASIC subroutine or function.

**Basic Structure:**

CSUB MyFunction integer, float

00000000 B085B480 6078AF00 687B6039

1AD368FB 687A3301 6839441A

End CSUB

The hexadecimal values represent compiled ARM machine code. You never edit these values directly - they are generated by compiling your C code using the ARM toolchain.

# Getting Started

## Required Tools

**ARM Toolchain:**

* macOS: brew install arm-none-eabi-gcc
* Linux: sudo apt-get install gcc-arm-none-eabi
* Windows: Download ARM GCC toolchain from ARM website

**Helper Tools (Recommended):**

The **picocalc\_csub\_helpers** toolkit from GitHub provides build scripts and examples that greatly simplify CSUB creation. Available at:

<https://github.com/jvanderberg/picocalc_csub_helpers>

# C Function Structure

## Basic Template

Every CSUB function follows this basic structure:

#include "PicoCFunctions.h"

long long your\_function\_name(void \*arg0, void \*arg1, ...) {

// Your C code here

return result;

}

## Key Components

* **Return type:** Always long long (64-bit integer)
* **Parameters:** Always void \* pointers to arguments
* **Header file:** Must include PicoCFunctions.h for MMBasic integration

# Complete Working Example

## Integer Power Function

This example implements an integer exponentiation function (calculating base raised to the power of exponent):

// pow\_int\_cf.c - Integer exponentiation

#include "PicoCFunctions.h"

long long pow\_int\_cf(void \*result\_ptr, void \*base\_ptr, void \*exp\_ptr) {

// Extract parameters from pointers

int base = \*(int\*)base\_ptr;

int exp = \*(int\*)exp\_ptr;

// Calculate result

int result = 1;

for(int i = 0; i < exp; i++) {

result \*= base;

}

// Write result back to first parameter

\*(int\*)result\_ptr = result;

return 0;

}

## Using in BASIC

Once compiled and converted to a CSUB, you can call it from your BASIC program:

' Define the CSUB

CSUB pow\_int integer, integer, integer

00000000 B085B480 6078AF00 687B6039

... (hex code continues) ...

End CSUB

' Call the function

DIM result%, base%, exp%

base% = 2

exp% = 10

pow\_int result%, base%, exp%

PRINT result% ' Prints 1024

# Compilation Process

## Compiling for RP2040 (Cortex-M0+)

For the original Raspberry Pi Pico with RP2040 processor:

arm-none-eabi-gcc -c -mthumb -mcpu=cortex-m0plus \

-O2 -fno-exceptions -ffunction-sections \

-o output.o your\_code.c

arm-none-eabi-ld -nostdlib -lgcc -T link.ld output.o -o output.elf

arm-none-eabi-objcopy -O binary output.elf output.bin

**Important:** RP2040 lacks hardware 64-bit multiply, so you must link with -lgcc to include the \_\_aeabi\_lmul helper function.

## Compiling for RP2350 (Cortex-M33)

For the Raspberry Pi Pico 2 with RP2350 processor:

arm-none-eabi-gcc -c -mthumb -mcpu=cortex-m33 \

-O2 -fno-exceptions -ffunction-sections \

-o output.o your\_code.c

## Converting to CSUB Format

After compilation, convert the binary to hexadecimal format for embedding in MMBasic:

# Using the helper script

./emit\_cfunction\_block.sh output.bin "function\_name integer, integer"

This generates the CSUB block ready to paste into your BASIC program.

# Parameter Handling

## Parameter Types

When declaring a CSUB, you specify the type of each parameter:

* **integer** - 64-bit signed integer
* **float** - Double precision floating point
* **string** - String variable

MMBasic will automatically convert types if possible or throw an error if incompatible types are provided.

## Accessing Parameters in C

Parameters are passed as void pointers. You must cast them to the appropriate type:

// For integers

int value = \*(int\*)param\_ptr;

// For floats

double value = \*(double\*)param\_ptr;

// For strings

char \*str = (char\*)param\_ptr;

// Writing results back (command-style)

\*(int\*)result\_ptr = calculated\_value;

# Best Practices

1. **Test in pure C first:** Debug your algorithm in a standard C environment before converting to a CSUB.
2. **Keep it simple:** Start with simple functions. Complex CSUBs are harder to debug.
3. **Use fixed-point math:** Often faster than floating-point on microcontrollers.
4. **Check alignment:** ARM processors prefer aligned memory access.
5. **Save working versions:** The hex CSUB code is discarded when saved to library. Keep source files.
6. **Handle errors carefully:** Buggy CSUBs can crash the system.
7. **Use the library:** Save bug-free CSUBs to the library for reuse.

# Advanced Topics

## Using the Helper Toolkit

The picocalc\_csub\_helpers toolkit simplifies the entire workflow:

# 1. Clone the repository

git clone https://github.com/jvanderberg/picocalc\_csub\_helpers

cd picocalc\_csub\_helpers

# 2. Set target processor

export CPU\_TARGET=rp2350 # or rp2040

# 3. Compile your function

./build.sh my\_func.c

# 4. Generate CSUB block

./emit\_cfunction\_block.sh my\_func.bin "my\_func integer, float"

## Saving to Library

Once your CSUB is working correctly, you can save it to the library:

LIBRARY SAVE

Benefits of using the library:

* CSUB is permanently available to all programs
* Hexadecimal version is discarded (only binary kept)
* Frees up program space
* Can be called from any program without re-embedding

**Warning:** Keep a copy of your source code and hex CSUB separately, as the library only stores the binary form.

## Accessing MMBasic Internals

The PicoCFunctions.h header provides access to:

* MMBasic firmware routines
* Firmware data structures
* Ability to call MMBasic functions from C
* I/O pin manipulation
* Memory management

**Note:** Ensure you use an updated PicoCFunctions.h header file from the latest firmware release for your device. The header file must match the firmware version.

# Troubleshooting

| **Problem** | **Solution** |
| --- | --- |
| CSUB crashes immediately | Check parameter types match declaration. Verify compilation for correct processor (RP2040 vs RP2350). |
| Incorrect results | Test C code separately. Check pointer dereferences and type casts. Verify you're writing results back correctly. |
| Compilation errors | Ensure PicoCFunctions.h is in include path. Check ARM toolchain is correctly installed. Verify CPU target matches your hardware. |
| MMBasic syntax error | Check CSUB declaration syntax. Ensure End CSUB is present. Verify hexadecimal format is correct (no missing lines). |
| Linker errors on RP2040 | Add -lgcc flag to linker command to include 64-bit multiply helper (\_\_aeabi\_lmul). |

# Performance Comparison

CSUBs provide significant performance improvements for computationally intensive tasks:

| **Task** | **Pure BASIC** | **CSUB** |
| --- | --- | --- |
| Mandelbrot 320x320 | ~70 seconds | **~10 seconds** |
| Tight math loops | Baseline | **7-10x faster** |
| Fixed-point operations | Baseline | **10-15x faster** |

These improvements make CSUBs ideal for graphics rendering, signal processing, cryptography, and other compute-intensive applications.

# Additional Resources

**Official Documentation:**

* PicoMite User Manual: [geoffg.net/picomite.html](http://geoffg.net/picomite.html)
* MMBasic Website: [mmbasic.com](http://mmbasic.com/)
* PicoMite Firmware Source: [github.com/UKTailwind/PicoMiteAllVersions](https://github.com/UKTailwind/PicoMiteAllVersions)

**Helper Tools:**

* CSUB Helper Toolkit: [github.com/jvanderberg/picocalc\_csub\_helpers](https://github.com/jvanderberg/picocalc_csub_helpers)

**Community Resources:**

* BackShed Forum: [thebackshed.com/forum/Microcontrollers](http://www.thebackshed.com/forum/Microcontrollers)
* FruitOfTheShed Wiki: [fruitoftheshed.com/wiki](https://fruitoftheshed.com/wiki/doku.php?id=mmbasic:csubs_and_cfunctions_-_creation)

**ARM Development:**

* ARM GCC Toolchain: [developer.arm.com/tools-and-software/open-source-software](https://developer.arm.com/tools-and-software/open-source-software)
* RP2040 Datasheet: [raspberrypi.com/documentation](https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html)
* RP2350 Datasheet: [raspberrypi.com/documentation](https://www.raspberrypi.com/documentation/microcontrollers/rp2350.html)

# Conclusion

CSUBs provide a powerful way to enhance MMBasic programs with native-speed performance for critical sections of code. By following the workflows and best practices outlined in this guide, you can create efficient, reliable CSUBs that dramatically improve the performance of your PicoMite applications.

The combination of MMBasic's ease of use for rapid development and CSUBs for performance-critical operations creates a powerful platform for embedded development on the Raspberry Pi Pico.

Whether you're building games, signal processing applications, or complex control systems, CSUBs allow you to achieve the performance you need without sacrificing the simplicity and flexibility of BASIC programming.

**Happy Coding!**