# GCC IFUNC and FMV Documentation

## Introduction

This document provides comprehensive guidelines on using GCC IFUNC (Indirect Functions) and FMV (Floating-Point Move Instructions) across different architectures. It includes an overview, usage instructions, architecture-specific details, best practices, and examples.

## IFUNC (Indirect Functions)

### Overview

IFUNC (Indirect Functions) is a feature in GCC that allows the selection of different function implementations at runtime based on various criteria such as CPU capabilities. This capability optimizes performance by selecting the most suitable function for the current execution environment.

### Usage

To use IFUNC, declare a function with the `ifunc` attribute and provide a resolver function that returns the appropriate function implementation.

\*\*Syntax:\*\*

\_\_attribute\_\_((ifunc("resolver\_function")))

\*\*Example:\*\*

#include <stdio.h>  
  
extern int foo(void) \_\_attribute\_\_((ifunc("foo\_resolver")));  
  
int foo\_impl1(void) {  
 return 1;  
}  
  
int foo\_impl2(void) {  
 return 2;  
}  
  
void \*foo\_resolver(void) {  
 // Example: Choose implementation based on some runtime condition  
 if (/\* condition \*/) {  
 return foo\_impl1;  
 } else {  
 return foo\_impl2;  
 }  
}  
  
int main(void) {  
 printf("%d\n", foo());  
 return 0;  
}

### Architecture-Specific Details

#### x86 Architecture

Support for IFUNC introduced in GCC version X.Y.

\*\*Usage:\*\*

#include <stdio.h>  
  
extern int foo(void) \_\_attribute\_\_((ifunc("foo\_resolver")));  
  
int foo\_sse(void) {  
 return 1;  
}  
  
int foo\_avx(void) {  
 return 2;  
}  
  
void \*foo\_resolver(void) {  
 // Choose implementation based on CPU capabilities  
 if (/\* check for AVX support \*/) {  
 return foo\_avx;  
 } else {  
 return foo\_sse;  
 }  
}  
  
int main(void) {  
 printf("%d\n", foo());  
 return 0;  
}

\*\*Performance Considerations:\*\* Ensure the resolver function is efficient as it can impact performance.

#### ARM Architecture

Support for IFUNC introduced in GCC version X.Y.

\*\*Usage:\*\*

#include <stdio.h>  
  
extern int foo(void) \_\_attribute\_\_((ifunc("foo\_resolver")));  
  
int foo\_neon(void) {  
 return 1;  
}  
  
int foo\_vfp(void) {  
 return 2;  
}  
  
void \*foo\_resolver(void) {  
 // Choose implementation based on CPU capabilities  
 if (/\* check for NEON support \*/) {  
 return foo\_neon;  
 } else {  
 return foo\_vfp;  
 }  
}  
  
int main(void) {  
 printf("%d\n", foo());  
 return 0;  
}

\*\*Performance Considerations:\*\* Optimize the resolver for minimal overhead.

### Best Practices

\*\*Resolver Efficiency:\*\* Ensure the resolver function is efficient to minimize performance overhead.

\*\*Performance-Critical Paths:\*\* Use IFUNC for performance-critical paths where different implementations can significantly improve performance based on runtime conditions.

### Examples

#### IFUNC Example for x86

#include <stdio.h>  
  
extern int foo(void) \_\_attribute\_\_((ifunc("foo\_resolver")));  
  
int foo\_sse(void) {  
 return 1;  
}  
  
int foo\_avx(void) {  
 return 2;  
}  
  
void \*foo\_resolver(void) {  
 // Choose implementation based on CPU capabilities  
 if (/\* check for AVX support \*/) {  
 return foo\_avx;  
 } else {  
 return foo\_sse;  
 }  
}  
  
int main(void) {  
 printf("%d\n", foo());  
 return 0;  
}

## FMV (Floating-Point Move Instructions)

### Overview

FMV (Floating-Point Move Instructions) are specific instructions related to floating-point operations in assembly languages. These instructions are crucial for high-performance computing tasks that involve floating-point calculations.

### Usage

The usage of FMV instructions varies by architecture. Here, we provide examples for common architectures such as x86 and ARM.

\*\*Syntax:\*\*

\*\*Example for x86\*\*

movaps %xmm0, %xmm1

\*\*Example for ARM\*\*

vmov.f32 s0, s1

### Architecture-Specific Details

#### x86 Architecture

\*\*Instructions:\*\* `movaps`, `movups`, etc.

\*\*Performance Considerations:\*\* Align data structures to natural boundaries to maximize performance.

#### ARM Architecture

\*\*Instructions:\*\* `vmov`, `vld1`, etc.

\*\*Performance Considerations:\*\* Minimize unnecessary floating-point moves to reduce overhead.

### Best Practices

\*\*Data Alignment:\*\* Align data structures to natural boundaries to maximize performance.

\*\*Reduce Overhead:\*\* Minimize unnecessary floating-point moves to reduce overhead.

### Examples

#### FMV Example for ARM

vmov.f32 s0, s1

## References

\*\*GCC Official Documentation:\*\* [GCC Documentation](https://gcc.gnu.org/onlinedocs/)

\*\*GCC Source Code Repository:\*\* [GCC Git Repository](https://gcc.gnu.org/git.html)

\*\*Community Forums:\*\* Engage with GCC mailing lists and forums for additional insights and support.

## Version History

\*\*Version 1.0:\*\* Initial release.

\*\*Version 1.1:\*\* Updated architecture-specific details and examples.