

# LLVM for Renesas RL78 User Manual

- Introduction
- Specific extensions for RL78
  - Command options
  - Extensions to the C language family
    - Predefined Target Macros
    - Keywords
    - Attributes
    - Pragmas
      - `#pragma interrupt [(interrupt-handler-name[(interrupt-specification [,...])])]`
      - `#pragma interrupt_brk [(interrupt-handler-name[(interrupt-specification [,...])])]`
      - `#pragma clang section [ section-type]=[ new-section-name]"`
      - `#pragma inline/noinline [(function-name [,...])]`
      - `#pragma saddr [(variable-name [,...])]`
      - `#pragma inline_asm [(function-name [,...])]`
      - `#pragma address [(variable-name=absolute-address [,...])]`
      - `#pragma callt [(function-name [,...])]`
  - Built-in / Embedded Functions
- Compatibility with GCC RL78
  - GCC RL78 target specific differences
  - Linker script compatibility
    - RL78 specific linker script requirements
  - Command Line options compatibility
    - Overall options
    - C language options
    - C++ language options
    - Language independent options
    - Warning options
    - C-only Warning Options
    - Debugging options
    - Preprocessor Options
    - Assembler options
    - Directory Options
    - Linker Options
    - Optimization options
    - Tool options
      - `llvm-objdump` compatibility with `rl78-elf-objdump`
  - Assembler compatibility
    - `.eqv` symbol, expression
    - `.fail` expression
    - `.hword`
    - `.lflags`
    - `.string8 "str"`, `.string16, 32, 64`
    - `.stabd`, `.stabn`, `.stabs`
    - `.mri val`
    - `.gnu_attribute tag,value`
    - `.func name[,label]`, `.endfunc`
    - `.vtable_entry table, offset`
    - `.vtable_inherit child, parent`

- [.loc\\_mark\\_labels enable](#)
- [Listing Directives: .list, .nolist, .eject, .psize lines, columns, .sbttl "subheading", .title "heading"](#)
- [C Language compatibility](#)
  - [Nested Functions](#)
  - [Constructing Calls](#)
  - [Pointer Arguments in Variadic Functions](#)
  - [Attributes](#)
  - [Vector Extensions](#)
  - [Miscellaneous Builtin functions](#)
  - [Bitfields](#)
  - [Extended Asm](#)
- [C++ Language compatibility](#)
  - [Extracting the function pointer from a bound pointer to member function](#)
  - [Namespace Association](#)
- [ABI compatibility with GCC RL78](#)
- [Compatibility with CCRL](#)
  - [Assembler compatibility](#)
    - [Operators and directives](#)
    - [Assembler generated symbols](#)
  - [ABI compatibility with CC-RL](#)
- [Libgen](#)

## Introduction

LLVM is an open source compiler just like GCC. It provides it's own frontend, Clang, for the C family of programming languages. In addition to language standards defined by ISO/IEC, Clang supports a broad variety of language extensions for compatibility with GCC and other compilers.

For information on how to use the toolchain please read the appropriate user manuals provided along with this file:

Inside the Doc directory of your toolchain installation folder:

- [Clang Compiler User's Manual — Clang 10 documentation](#)

Inside the Tools Command Guide directory:

- [ClangFormat — Clang 10 documentation](#)
- [LLVM Command Guide — LLVM 10 documentation](#)
- [llvm-addr2line - a drop-in replacement for addr2line — LLVM 10 documentation](#)
- [llvm-ar - LLVM archiver — LLVM 10 documentation](#)
- [llvm-cxxfilt - LLVM symbol name demangler — LLVM 10 documentation](#)
- [llvm-dwarfdump - dump and verify DWARF debug information — LLVM 10 documentation](#)
- [llvm-nm - list LLVM bitcode and object file's symbol table — LLVM 10 documentation](#)
- [llvm-objcopy - object copying and editing tool — LLVM 10 documentation](#)
- [llvm-objdump - LLVM's object file dumper — LLVM 10 documentation](#)
- [llvm-ranlib - generates an archive index — LLVM 10 documentation](#)
- [llvm-readelf - GNU-style LLVM Object Reader — LLVM 10 documentation](#)
- [llvm-readobj - LLVM Object Reader — LLVM 10 documentation](#)
- [llvm-size - print size information — LLVM 10 documentation](#)
- [llvm-strings - print strings — LLVM 10 documentation](#)
- [llvm-strip - object stripping tool — LLVM 10 documentation](#)
- [llvm-symbolizer - convert addresses into source code locations — LLVM 10 documentation](#)

OR visit the online version of the manuals:

- <https://releases.llvm.org/10.0.0/tools/clang/docs/UsersManual.html>
- <https://releases.llvm.org/10.0.0/tools/clang/docs/ClangFormat.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/index.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-symbolizer.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-dwarfdump.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-readobj.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-addr2line.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-ar.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-cxxfilt.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-nm.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-objcopy.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-objdump.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-ranlib.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-readelf.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-size.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-objdump.html>
- <https://releases.llvm.org/10.0.0/docs/CommandGuide/llvm-strip.html>

The majority of the features are described in the User manual above are available for RL78 as well. However some features presented in the above manuals are not available for RL78 in which case the compiler will signal this the following way:

clang: error: unsupported option '...' for target 'rl78'

## Specific extensions for RL78

### Command options

#### **-fsim**

Links in additional target libraries to support operation within GDB simulator.

#### **-mcpu=S1**

#### **-mcpu=S2**

#### **-mcpu=S3**

Specifies the RL78 core to target, the default is the S3 core.

The S2 core does not have multiply or divide instructions, instead it uses a hardware peripheral for these operations and the S1 core lacks hardware multiplication altogether(except the MULU instruction).

Thus specifying -mcpu=S2 or -mcpu=S1 disables the use of the multiplication instructions (MULHU, MULH, DIVHU, DIVWU, MACHU, MACH) and software routines will be used instead.

#### **-frenesas-extensions**

Enables some CCRL extensions (see following sections) and specifications:

- renames sections .rodata/.frodta to .const/.constf
- when using #pragma clang section to specify custom section names, it appends \_n or \_f suffixes to the section names, depending on the address space used
- enables CC-RL specific pragmas and modifies the syntax of others, see section 2.2.4 Pragmas
- this option cannot be used together with the -save-temps option

#### **-frenesas-vaarg**

Causes the promotion of \_\_near pointers to \_\_far pointers when they are passed as variable arguments.

In a future release, this option will be merged with the -frenesas-extensions option.

When omitted, pointers are passed without promotion.

**-m64bit-doubles**

Makes the double data type to be represented on 64 bits, the default being 32 bits.

**-mda-disable**

Disables the generation of division/multiplication and multiply-accumulate unit (S2 core only).

**-mfar-data**

Global, non-const variables will be allocated to the far address space. Explicit address space qualifiers or the `__saddr` qualifier override this.

Data pointers that point to non-const variables, without explicit address space qualifiers, will be far pointers.

If not specified, all variables without explicit address space qualifiers will be allocated to the near address space.

**-mnear-data**

Global, non-const variables will be allocated to the near address space. Explicit address space qualifiers or the `__saddr` qualifier override this.

This is the default, even if this option is omitted.

**-mfar-rom**

Global, constant data variables without explicit address space qualifiers will be allocated to the far address space.

Pointers without explicit address space qualifiers that point to such variables will be far pointers.

If not specified, variables and pointers without explicit address space qualifiers will be near.

Support for far rom is still limited in the toolchain libraries. The following functions had their far rom equivalents added: `atof`, `atoff`, `atoi`, `atol`, `atoll`, `bsearch`, `gets`, `memchr`, `memcmp`, `memcpy`, `memmove`, `memset`, `perror`, `puts`, `qsort`, `strcat`, `strchr`, `strcmp`, `strcpy`, `strcspn`, `strlen`, `strncat`, `strncmp`, `strncpy`, `strpbrk`, `strrchr`, `strspn`, `strstr`, `strtod`, `strtof`, `strtoimax`, `strtol`, `strtold`, `strtoll`, `strtoul`, `strtoull`, `strtoumax`

Users can include `<stdlib.h>`, `<stdio.h>` or `<string.h>` and call these far versions directly or by using the `-mfar-rom` command-line option, calls to the near versions of the functions will be replaced with calls to the far versions.

For example, if we have the following source code:

```
#include<string.h>
void foo(void *d, const void *s, size_t n){
    memcpy(d,s,n);
}

void bar(void    far *d, const void __far *s, size_t n){
    __COM_memcpy_ff(d,s,n);
}
```

`clang -c test.c` will generate the following assembly:

```
foo:
    call !_memcpy
    ret
bar:
    push ax
    xch a, h
    mov a, x
    xch a, h
    mov l, a
    movw ax, [sp+6]
    movw [sp+0], ax
    xch a, x
    mov a, h
    xch a, x
    mov a, l
    call !!__COM_memcpy_ff
```

```

    addw sp, #2
    ret

```

calling `clang -c test.c -mfar-rom` will result in the following code

```

foo:
    subw sp, #4
    xch a, c
    mov [sp+2], a
    xch a, c
    push de
    pop bc
    movw de, ax
    movw ax, [sp+8]
    movw [sp+0], ax
    movw ax, de
    cmpw ax, #0
    bz $.LBB0_1
    movw hl, #15
    br $.LBB0_3
.LBB0_1:
    movw hl, #0
.LBB0_3:
    mov a, [sp+2]
    mov x, a
    mov a, l
    call !! __COM_memcpy_ff
    addw sp, #4
    ret

bar:
    push ax
    xch a, h
    mov a, x
    xch a, h
    mov l, a
    movw ax, [sp+6]
    movw [sp+0], ax
    xch a, x
    mov a, h
    xch a, x
    mov a, l
    call !! __COM_memcpy_ff
    addw sp, #2
    ret

```

Please note that `bar` remained unchanged and `foo` address space casts it's first parameter before calling the far version of the function.

The signatures of the far rom function versions can be seen below:

```

double __COM_atof_f(const char __far *nptr);
float __COM_atoff_f(const char __far *nptr);
int __COM_atoi_f(const char __far *nptr);
long int __COM_atol_f(const char __far *nptr);

```

```

long long int _COM_atoll_f(const char __far *nptr);
void __far * _COM_bsearch_f(const void __far *key, const void __far *base, size_t nmemb, size_t size, int (*compar)
(const void __far *, const void __far *));
char __far * _COM_gets_f(char __far *s);
void __far * _COM_memchr_f(const void __far *s, int c, size_t n);
int _COM_memcmp_ff(const void __far *s1, const void __far *s2, size_t n);
void __far * _COM_memcpy_ff(void __far * __restrict s1, const void __far * __restrict s2, size_t n);
void __far * _COM_memmove_ff(void __far *s1, const void __far *s2, size_t n);
void __far * _COM_memset_f(void __far *s, int c, size_t n);
void _COM_perror_f(const char __far *s);
int _COM_puts_f(const char __far *s);
void _COM_qsort_f(void __far *base, size_t nmemb, size_t size, int (*compar)(const void __far *, const void __far *));
char __far * _COM_strcat_ff(char __far * __restrict s1, const char __far * __restrict s2);
char __far * _COM_strchr_f(const char __far *s, int c);
int _COM_strcmp_ff(const char __far *s1, const char __far *s2);
char __far * _COM_strcpy_ff(char __far * __restrict s1, const char __far * __restrict s2);
size_t _COM_strcspn_ff(const char __far *s1, const char __far *s2);
size_t _COM_strlen_f(const char __far *s);
char __far * _COM_strncat_ff(char __far * __restrict s1, const char __far * __restrict s2, size_t n);
int _COM_strncmp_ff(const char __far *s1, const char __far *s2, size_t n);
char __far * _COM_strncpy_ff(char __far * __restrict s1, const char __far * __restrict s2, size_t n);
char __far * _COM_strpbrk_ff(const char __far *s1, const char __far *s2);
char __far * _COM_strrchr_f(const char __far *s, int c);
size_t _COM_strspn_ff(const char __far *s1, const char __far *s2);
char __far * _COM_strstr_ff(const char __far *s1, const char __far *s2);
double _COM_strtod_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr);
float _COM_strtof_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr);
intmax_t _COM_strtoimax_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int base);
long int _COM_strtol_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int base);
long double _COM_strtold_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr);
long long int _COM_strtoll_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int base);
unsigned long int _COM_strtoul_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int base);
unsigned long long int _COM_strtoull_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int
base);
uintmax_t _COM_strtoumax_ff(const char __far * __restrict nptr, char __far * __far * __restrict endptr, int base);

```

### **-mfar-code**

This option changes the allocation of all functions, without an explicit address space, to `__far`.  
Pointers to functions that have no address space specifiers will also become `__far` pointers.

Functions qualified explicitly or implicitly (as a result of this option) with `__far` will be placed in the `.textf` section, which needs to be present in the linkerfile that is used during linking.

This option also disables the usage of the PLT, which was used to handle calls and jumps that are allocated over the 64KB limit.

When using the option, the following considerations should be made:

- The linkerfile may require changes: there's no longer a need for the `.plt` section and the `.textf` section should be specified.
- Since `__far` function pointers are 4 bytes, any code relying on function pointers being 2 byte long will require adjustments (for example startup code that iterates over an initializer function list).
- When writing assembly code, to ensure correct linking, the `.textf` section should be declared with the correct flags, as seen below:

```
.section .textf,#alloc,#execinstr
.global _foo
.type _foo,@function
_foo:
    ;rest of the function
```

Currently the `-mfar-code` option is only supported when compiling and linking C files, C++ is not supported.

When using the `-mfar-code` option, function typedefs can't have address space specifiers, nor can an address space specifier applied later to them. This will be fixed in the next release.

#### **-mnear-code**

Default function pointers are near pointers and functions are allocated to the near address space.

#### **-mllvm -use-section-attribute-in-outlined-functions**

Instruct the compiler to only consider outlining of functions that are located in the same section.

```
void Func1() __attribute__((section("SEC1")));
void Func2() __attribute__((section("SEC1")));
void Func3();
```

Using `-mllvm -use-section-attribute-in-outlined-functions` will result in `Func1` and `Func2` being considered for common code outlining.

`Func3` will not be considered since it is in a different section.

As of version 202306, this is the default.

#### **-Wl,-stride-dsp-memory-area**

Instructs the linker to not allocate in the DSP memory area: `[0xFD800, 0xFF000)`.

When not specified or with the `-no-stride-dsp-memory-area` option specified, the linker will allocate in the given memory range too, if needed.

## Extensions to the C language family

### Predefined Target Macros

#### **\_\_ELF\_\_**

Defined: Always

#### **\_\_RL78\_\_**

Defined: Always

#### **\_\_RL78\_S1\_\_**

Defined: if `-mcpu=S1` is specified

#### **\_\_RL78\_S2\_\_**

Defined: if `-mcpu=S2` is specified

#### **\_\_RL78\_S3\_\_**

Defined: if `-mcpu=S3` is specified or the `-mcpu` option is omitted

#### **\_\_RL78\_32BIT\_DOUBLES\_\_**

Defined: if `-m64bit-doubles` is omitted

**\_\_RL78\_64BIT\_DOUBLES\_\_**

Defined: if -m64bit-doubles is specified

**\_\_FAR\_ROM\_\_**

Defined: if -mfar-rom is specified

**\_\_RL78\_SMALL\_\_**

Defined: if -mnear-code option is specified implicitly or explicitly. For details, see -mnear-code.

**\_\_RL78\_MEDIUM\_\_**

Defined: if -mfar-code option is specified implicitly or explicitly. For details, see -mfar-code.

**\_\_CCRL\_\_**

Defined: if -frenesas-extensions is specified

**\_\_CCRL**

Defined: if -frenesas-extensions is specified

**\_\_DBL4**

Defined: if -frenesas-extensions is specified and -m64bit-doubles is omitted

**\_\_DBL8**

Defined: if -frenesas-extensions and -m64bit-doubles is specified

**\_\_SCHAR**

Defined: if -frenesas-extensions is specified and -funsigned-char is omitted

**\_\_UCHAR**

Defined: if -frenesas-extensions and -funsigned-char is specified

**\_\_SBIT**

Defined: if -frenesas-extensions is specified

**\_\_MDA\_ENABLED\_\_**

Defined: if -mpcu=S2 is specified and -mdisable-mda is not specified

**\_\_MDA\_DISABLED\_\_**

Defined: if both -mpcu=S2 and -mdisable-mda is specified

## Keywords

**\_\_saddr**

This specifies placement of the corresponding variable in the SADDR area, which can be accessed more efficiently than the default memory region.

OBS. Available only when -frenesas-extensions option is passed to the compiler.

**\_\_callt**

This specifies that the function should be called with the callt instruction.

OBS. Available only when -frenesas-extensions option is passed to the compiler.

**\_\_near**

Variables qualified with it are accessed with 16-bit pointers. This is the default behavior, unless -mfar-data or -mfar-rom is used to override it.

**\_\_far**

Variables qualified with it are accessed with 32-bit pointers (20-bit addresses) rather than the default 16-bit addresses.

Non-far variables are assumed to appear in the topmost 64 KiB of the address space.



**\_\_inline**

This notifies the compiler of an inline function, which could be expanded inline at the location from which the function is called.

It behaves according to C99.

**\_\_sectop/\_\_secend**

Section address operators:

\_\_sectop: returns the start address of the given section

\_\_secend: returns the end address + 1 of the given section

Usage:

\_\_sectop("section-name")

\_\_secend("section-name")

OBS. Available only when -frenesas-extensions option is passed to the compiler.

## Attributes

**saddr**

USAGE: Variables

SYNTAX: \_\_attribute\_\_((saddr))

DESCRIPTION: This specifies placement of the corresponding variable in the SADDR area, which can be accessed more efficiently than the default memory region.

**callt**

USAGE: Functions

SYNTAX: \_\_attribute\_\_((callt))

DESCRIPTION: This specifies that the function should be called with the callt instruction.

**naked**

USAGE: Functions

SYNTAX: \_\_attribute\_\_((naked))

DESCRIPTION: This attribute allows the compiler to construct the requisite function declaration, while allowing the body of the function to be assembly code.

The specified function will not have prologue/epilogue sequences generated by the compiler.

Only basic asm statements can safely be included in naked functions.

While using extended asm or a mixture of basic asm and C code may appear to work, they cannot be depended upon to work reliably and are not supported.

**interrupt**

USAGE: Functions

SYNTAX: \_\_attribute\_\_((interrupt))

\_\_attribute\_\_((interrupt([ *interrupt specification* ])))

DESCRIPTION: Indicates that the specified function is a hardware interrupt handler.

The compiler generates function entry and exit sequences suitable for use in an interrupt handler when this attribute is present.

See #pragma interrupt for details on the interrupt specification.

**brk\_interrupt**

USAGE: Functions

SYNTAX: \_\_attribute\_\_((brk\_interrupt))

\_\_attribute\_\_((brk\_interrupt([ *interrupt specification* ])))

DESCRIPTION: Indicates that the specified function is a software interrupt handler, used for handlers with the BRK opcode (i.e. those that must end with RETB instead of RETI).

See #pragma brk\_interrupt for details on the interrupt specification.

## Pragmas

### **#pragma interrupt [(*interrupt-handler-name*[(*interrupt-specification* [,...])])]**

#### DESCRIPTION:

- Enabled when specifying `-frenesas-extensions`
- Indicates that the specified function is a hardware interrupt handler.
- A code for returning with RETI is generated with the target function definition.
- A stack or a register bank can be specified as the area for saving general registers.
- The interrupt handler definition is output to the `.text` or `.textf` section in the same way as normal function definitions.
- When no register is used or no function is called in an interrupt handler, the instruction for switching register banks is not output even if register bank switching is specified in `#pragma interrupt`.

#### USAGE:

The interrupt specifications can include the following:

- **vect=*address***  
When specified, `___vector[address in decimal]` label will be emitted at the location of the interrupt handler function.  
The emitted label can be referenced in the linkerscript, to be able to place the handler at the correct memory location.  
*Address*: Binary, octal, decimal, or hexadecimal constant  
Only an even value within the range from 0x0 to 0x7c can be specified.  
ex: `vect=0x12` results in emitting `___vector18`
- **bank={*RB0*|*RB1*|*RB2*|*RB3*}**  
Register bank for use by the interrupt handler.  
Changing the register bank makes it unnecessary to save the values of general registers on the stack.  
Note that ES and CS are saved on the stack.  
When this setting is omitted, the general registers are saved on the stack.
- **enable={*true*|*false*}**  
If set to true, enables nested interrupts at the entry to a function, resulting in the EI instruction being generated before the register saving code.  
If set to false or omitted, the EI instruction is not generated.

#### EXAMPLES:

```
#pragma interrupt test0
#pragma interrupt test1(vect=0x12,bank=RB2,enable=true)

void test0(void) {
/*Interrupt processing*/
}

void test1(void) {
/*Interrupt processing*/
}
```

```
#pragma interrupt_brk [(interrupt-handler-name[(interrupt-specification[,...])])]
```

DESCRIPTION:

- Enabled when specifying `-frenesas-extensions`
- Indicates that the specified function is a software interrupt handler.
- A code for returning with RETB is generated with the target function definition.
- A stack or a register bank can be specified as the area for saving general registers.
- The interrupt handler definition is output to the `.text` or `.textf` section in the same way as normal function definitions.
- When no register is used or no function is called in an interrupt handler, the instruction for switching register banks is not output even if register bank switching is specified in `#pragma interrupt`.

USAGE:

The interrupt specifications can include the following:

- **bank={RB0|RB1|RB2|RB3}**  
 Register bank for use by the interrupt handler.  
 Changing the register bank makes it unnecessary to save the values of general registers on the stack.  
 Note that ES and CS are saved on the stack.  
 When this setting is omitted, the general registers are saved on the stack.
- **enable={true|false}**  
 If set to true, enables nested interrupts at the entry to a function, resulting in the EI instruction being generated before the register saving code.  
 If set to false or omitted, the EI instruction is not generated.

EXAMPLE:

```
#pragma interrupt_brk test2(bank=RB3,enable=false)
void test2(void) {
/*Interrupt processing*/
}
```

```
#pragma clang section [ section-type]="[ new-section-name]"
```

DESCRIPTION:

- Assign a new section name that differs from the default name to a specified set of data or functions allows this set of data or functions to be allocated to an address separate from that for other data or functions, and special processing can be applied in section units.

USAGE:

Allowed section types are: `[bss|data|rodata|text]`

Default names of the sections by type are:

- `bss = ".bss"`
- `data = ".data"`
- `rodata = ".rodata" or ".const"` if `-frenesas-extensions` is specified
- `text = ".text"`

**Note:** Currently specifying **#pragma clang section** without `section type=name` pairs does not reset the section names to their default.

When using -frenesas-extensions, the custom section names will have a suffix added to them, based on the following rules:

- near section (.text, .const, .data, .bss): new section name + "\_n"
- far section (.textf, .constf, .dataf, .bssf): new section name + "\_f"
- saddr section (.sdata, .sbss): new section name + "\_s"

**Note:** Currently explicitly resetting section names to their defaults will not work when using -frenesas-extensions

EXAMPLE:

```
const int c0;
int b0;
__far int b0_far;
int d0 = 3;

#pragma clang section rodata="C"
const int c1;
#pragma clang section bss="B" data="D" text="T"
int b1;
int d = 2;
#pragma clang section bss=".bss" data=".data" rodata=".rodata" text=".text" //reset default section names
const int c2;
int b2;
int d2 = 3;
```

**#pragma inline/noinline** [(**function-name** [,...][**]**)]

DESCRIPTION:

- Enabled when specifying -frenesas-extensions
- #pragma inline declares a function to be expanded inline.
- #pragma noinline disables inline expansion when using optimization -O3

EXAMPLE:

```
#pragma inline foo
int foo(int b) {
/* instructions will be inlined */
}

#pragma noinline bar
int bar(int b) {
/* instructions will not be inlined when using -O3 */
}
```

**#pragma saddr** [(**variable-name** [,...][**]**)]

DESCRIPTION:

- Enabled when specifying -frenesas-extensions
- Notifies the compiler of a variable that is to be assigned to the saddr area.
- Initialized variables are allocated to the .sdata section.
- Uninitialized variables are allocated to the .sbss section.

EXAMPLE:

```
#pragma saddr saddr_var
extern int saddr_var;
void func(void)
{
    saddr_var = 0;
}
```

**#pragma inline\_asm [(]function-name [,...][)]**

DESCRIPTION:

- Enabled when specifying -frenesas-extensions
- Allows usage of assembly instructions in the given function.
- **Note: In the current release it will not inline the assembly instructions, contrary to it's CC-RL counterpart.**

EXAMPLE:

```
int a;
#pragma inline_asm func
void func(int x)
{
    movw !_a, ax
}

int main(void) {
    func(3);
    return a;
}
```

**#pragma address [(]variable-name=absolute-address[,...][)]**

DESCRIPTION:

- Enabled when specifying -frenesas-extensions
- Allows the specification of the absolute address where the variable will be allocated.

EXAMPLE:

```
#pragma address (io=0x0ffe00)
int io; //io is allocated to address 0x0ffe00

void func(void){
    io = 12;
}

int main() {
    func();
    return io;
}
```

**#pragma callt [(]function-name[,...][)]**

DESCRIPTION:

- Enabled when specifying -frenesas-extensions
- Same as the \_\_callt keyword or attribute, this specifies that the function should be called with the callt instruction.

EXAMPLE:

```
#pragma callt callt_func1
extern void callt_func1(void);
void func1(void)
{
    callt_func1();
}
```

## Built-in / Embedded Functions

**unsigned int \_\_mulu(unsigned char x, unsigned char y);**

Executes 8-bit unsigned multiplication between x and y and returns a 16-bit result.

**unsigned long \_\_mului(unsigned int x, unsigned int y);**

Executes 16-bit unsigned multiplication between x and y and returns a 32-bit result.

**signed long \_\_mulsi(signed int x, signed int y);**

Executes 16-bit signed multiplication between x and y and returns a 32-bit result.

**unsigned int \_\_divui(unsigned int x, unsigned char y);**

Executes unsigned division between x and y and returns a 16-bit result.

When divisor y is 0, 0xFFFF is returned.

**unsigned long \_\_divul(unsigned long x, unsigned int y);**

Executes unsigned division between x and y and returns a 32-bit result.

When divisor y is 0, 0xFFFFFFFF is returned.

**unsigned char \_\_remui(unsigned int x, unsigned char y);**

Executes unsigned remainder operation between x and y and returns a 8-bit result.

When divisor y is 0, the lower-order 8 bits of dividend x are returned.

**void \_\_builtin\_rl78\_mov1 (char \*x, char b1, char y, char b2);**

Moves bit b2 of y to bit b1 of x.

**void \_\_builtin\_rl78\_and1 (char \*x, char b1, char y, char b2);**

Executes AND between bit b2 of y and bit b1 of x, storing the result in bit b1 of x.

**void \_\_builtin\_rl78\_or1 (char \*x, char b1, char y, char b2);**

Executes OR between bit b2 of y and bit b1 of x, storing the result in bit b1 of x.

**void \_\_builtin\_rl78\_xor1 (char \*x, char b1, char y, char b2);**

Executes XOR between bit b2 of y and bit b1 of x, storing the result in bit b1 of x.

**void \_\_builtin\_rl78\_set1 (char \*x, char b);**

Sets bit b of x.

**void \_\_builtin\_rl78\_clr1 (char \*x, char b);**

Clears bit b of x.

**void \_\_builtin\_rl78\_not1 (char \*x, char b);**

Flips bit b of x.

**char \_\_builtin\_rl78\_ror1 (char x);**

Rotates x to the right once.

**char \_\_builtin\_rl78\_rol1 (char x);**

Rotates x to the left once.

**void \_\_builtin\_rl78\_ei ();**  
**\_\_EI();**

Execute the EI instruction.

**void \_\_builtin\_rl78\_di ();**  
**\_\_DI();**

Execute the DI instruction.

**char \_\_builtin\_rl78\_pswie ();**

Return the value of the Interrupt enable flag (IE) from the Program status word (PSW) register.

**void \_\_builtin\_rl78\_setpswisp (char);**

Set the values of the In-service priority flags (ISP0 and ISP1) in the Program status word (PSW) register to the argument value.

**char \_\_builtin\_rl78\_getpswisp ();**

Return the values of the In-service priority flags (ISP0 and ISP1) from the Program status word (PSW) register.

**void \_\_halt();**

Execute the HALT instruction.

**void \_\_stop();**

Execute the STOP instruction.

**void \_\_brk();**

Execute the BRK instruction.

**void \_\_nop();**

Execute the NOP instruction.

**unsigned char \_\_rolb(unsigned char x, unsigned char y);**

Rotates x to the left y times assuming that x has a size of eight bits

**unsigned char \_\_rorb(unsigned char x, unsigned char y);**

Rotates x to the right y times assuming that x has a size of eight bits.

**unsigned int \_\_rolw(unsigned int x, unsigned char y);**

Rotates x to the left y times assuming that x has a size of 16 bits.

**unsigned int \_\_rorw(unsigned int x, unsigned char y);**

Rotates x to the right y times assuming that x has a size of 16 bits.

**unsigned long long \_\_mulul(unsigned long x, unsigned long y);**

Executes signed multiplication between (unsigned long)x and (unsigned long)y and returns a 64-bit result.

**signed long long \_\_mulsl(signed long x, signed long y);**

Executes signed multiplication between (signed long)x and (signed long)y and returns a 64-bit result.

**unsigned int \_\_remul(unsigned long x, unsigned int y);**

Executes unsigned remainder operation between x and y and returns a 16-bit result.

**unsigned long \_\_macui(unsigned int x, unsigned int y, unsigned long z);**

Executes unsigned multiply-accumulate operation (unsigned int) x \* (unsigned int) y + z, and returns a 32-bit result.

**signed long \_\_macsi(signed int x, signed int y, signed long z);**

Executes signed multiply-accumulate operation (signed int) x \* (signed int) y + z, and returns a 32-bit result.

## Compatibility with GCC RL78

Clang\LLVM is designed as straightforward replacement for GCC. It has already successfully replaced GCC in quite a few projects.

The compatibility between LLVM and GCC can be seen in all aspects command line options, C/C++ extensions, assembler directives.

That being said there still are some differences between LLVM and GCC which we will discuss in detail in the following sections.

For further information on the compatibility between LLVM and GCC please check:

<https://clang.llvm.org/compatibility.html>

## GCC RL78 target specific differences

No equivalent options for `-msave-cs-in-interrupts`, `-msave-mduc-in-interrupts` and `-muse-es`. Just like CCRL, LLVM we can detect automatically which registers are used in the function so no need for such options to save those registers all the time regardless is used/not used.

No exact equivalent option for `-mes0`. `-mfar-rom` is the similar option in LLVM RL78 which is implemented in a similar way to CCRL's `-far_rom` option.

No `-mg10`, `-mg13`, `-mg14`, `-mcpu=g10`, `-mcpu=g13`, `-mcpu=g14` alias options present. LLVM only provides the correct core names for the `-mcpu` option (S1, S2, S3). The same is true for the corresponding predefined macros `__RL78_G10__`, `__RL78_G13__`, `__RL78_G14__` which are not defined by LLVM RL78 instead the correct macros (`__RL78_S1__`, `__RL78_S2__`, `__RL78_S3__`) should be used.

## Linker script compatibility

LLD implements a large subset of the GNU ld linker script notation. The LLD implementation policy is to implement linker script features as they are documented in the ld [manual](#). It is consider a bug if the lld implementation does not agree with the manual and it is not mentioned as an exception.

For further information on the compatibility between LLD and GNU LD linker script support please check:

[https://lld.llvm.org/ELF/linker\\_script.html](https://lld.llvm.org/ELF/linker_script.html)

## RL78 specific linker script requirements

`__data` symbol indicates the start of RAM, must be 2-byte aligned. If `__data` is not defined, the linker will consider that the start of RAM is at `0xfef00`.

`__end` symbol indicates the start of heap, must be 128-byte aligned.

## Command Line options compatibility

The majority of Clang options have the same name and functionality as they do in GCC. On top of those, in order to facilitate migration from GCC, Clang has provided dummy implementations for those options: some GCC options are ignored silently while in case other GCC options clang will issue a warning (similar to "warning: optimization flag '...' is not supported [-Wignored-optimization-argument]")



Not all GCC options have dummy implementations in Clang so in some cases Clang will return an error for unrecognized options:

clang.exe: error: unknown argument: ',,,'

## Overall options

### *-pass-exit-codes*

DESCRIPTION: Normally the gcc program exits with the code of 1 if any phase of the compiler returns a non-success return code. If you specify '-pass-exit-codes', the gcc program instead returns with the numerically highest error produced by any phase returning an error indication. The C, C++, and Fortran front ends return 4 if an internal compiler error is encountered.

ALTERNATIVE: Not needed. Clang has different exit codes in case of error anyway, and tools like make only care if the exit code is 0 or different from 0.

### *-wrapper*

DESCRIPTION: Invoke all subcommands under a wrapper program. The name of the wrapper program and its parameters are passed as a comma separated list.

ALTERNATIVE: Not needed. In a wrapper program is needed it can be written without any help from clang.

### *--target-help*

DESCRIPTION: Print (on the standard output) a description of target-specific command-line options for each tool. For some targets extra target-specific information may also be printed.

ALTERNATIVE: Not needed. clang has different help.

## C language options

### *-aux-info filename*

DESCRIPTION: Output to the given filename prototyped declarations for all functions declared and/or defined in a translation unit, including those in header files. This option is silently ignored in any language other than C.

ALTERNATIVE: Not needed. The option only outputs information in a file and is also ignored in any language other than C.

### *-fallow-parameterless-variadic-functions*

DESCRIPTION: Accept variadic functions without named parameters. Although it is possible to define such a function, this is not very useful as it is not possible to read the arguments. This is only supported for C as this construct is allowed by C++.

ALTERNATIVE: Not needed. As the description says this option is not useful in GCC either.

### *-fplan9-extensions*

DESCRIPTION: Accept some non-standard constructs used in Plan 9 code. This enables '-fms-extensions', permits passing pointers to structures with anonymous fields to functions that expect pointers to elements of the type of the field, and permits referring to anonymous fields declared using a typedef.

ALTERNATIVE: -fms-extensions + code modifications for the rest of non-standard constructs.

### *-traditional*

DESCRIPTION: Formerly, this option caused GCC to attempt to emulate a pre-standard C compiler. It is now only supported with the '-E' switch. The preprocessor continues to support a pre-standard mode.

ALTERNATIVE: Not needed. There's no support for pre-standard C in clang.

### *-fcond-mismatch*

DESCRIPTION: Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void. This option is not supported for C++.

ALTERNATIVE: code modifications for non-standard constructs.

## C++ language options

### *-fabi-version:*

DESCRIPTION: Use version n of the C++ ABI.

ALTERNATIVE: Not needed, not applicable LLVM has a different C++ library.

### *-fno-enforce-eh-specs:*

DESCRIPTION: Don't generate code to check for violation of exception specifications at run time. This option violates the C++ standard, but may be useful for reducing code size in production builds, much like defining 'NDEBUG'. This does not give user code permission to throw exceptions in violation of the exception specifications; the compiler still optimizes based on the specifications, so throwing an unexpected exception results in undefined behavior at run time.

ALTERNATIVE: Not needed, it violates C++ standard and can result in undefined behavior, code size reduction is achieved through optimizations.

### *-fno-for-scope:*

DESCRIPTION: If '*-ffor-scope*' is specified, the scope of variables declared in a *for-init-statement* is limited to the 'for' loop itself, as specified by the C++ standard. If '*-fno-for-scope*' is specified, the scope of variables declared in a *for-init-statement* extends to the end of the enclosing scope, as was the case in old versions of G++, and other (traditional) implementations of C++.

Alternative: No alternative, source code needs to be modified.

### *-fno-implicit-inline-templates:*

DESCRIPTION: Don't emit code for implicit instantiations of inline templates, either. The default is to handle inlines differently so that compiles with and without optimization need the same set of explicit instantiations.

ALTERNATIVE: Not needed, clang a different way of implementing templates.

### *-fno-nonansi-builtins:*

DESCRIPTION: Disable built-in declarations of functions that are not mandated by ANSI/ISO C. These include `ffs`, `alloca`, `_exit`, `index`, `bzero`, `confj`, and other related functions.

ALTERNATIVE: Not needed. Only affects generation of warning messages.

### *-fnthrow-opt:*

DESCRIPTION: Treat a `throw()` exception specification as if it were a `noexcept` specification to reduce or eliminate the text size overhead relative to a function with no exception specification. If the function has local variables of types with non-trivial destructors, the exception specification actually makes the function smaller because the EH cleanups for those variables can be optimized away. The semantic effect is that an exception thrown out of a function with such an exception specification results in a call to terminate rather than unexpected.

ALTERNATIVE: Affects standard exception behavior, if code size is an issue due to exceptions exceptions can be disabled altogether.

### *-fno-optional-diags:*

DESCRIPTION: Disable diagnostics that the standard says a compiler does not need to issue. Currently, the only such diagnostic issued by G++ is the one for a name having multiple meanings within a class.

ALTERNATIVE: Not needed, this is a option which controls warnings which do not affect output program in any way. Clang has a different warning messaging system with different options.

`-fno-pretty-templates:`

DESCRIPTION: When an error message refers to a specialization of a function template, the compiler normally prints the signature of the template followed by the template arguments and any typedefs or typenames in the signature (e.g. `void f(T) [with T = int]` rather than `void f(int)`) so that it's clear which template is involved. When an error message refers to a specialization of a class template, the compiler omits any template arguments that match the default template arguments for that template. If either of these behaviors make it harder to understand the error message rather than easier, you can use `'-fno-pretty-templates'` to disable them.

ALTERNATIVE: Not needed. Clang has a different messaging system.

`-frepo:`

DESCRIPTION: Enable automatic template instantiation at link time. This option also implies `'-fno-implicit-templates'`.

ALTERNATIVE: Not needed. Clang\LLVM has a different template implementation.

`-fno-weak:`

DESCRIPTION: Do not use weak symbol support, even if it is provided by the linker. By default, G++ uses weak symbols if they are available. This option exists only for testing, and should not be used by end-users; it results in inferior code and has no benefits. This option may be removed in a future release of G++.

ALTERNATIVE: Not needed. As stated above the option has no benefits.

`-fvtable-verify, -fvtv-counts, -fvtv-debug:`

DESCRIPTION: Debugging flags related to vtable verification.

ALTERNATIVE: Not needed. Useful for debugging GCC.

`-fext-numeric-literals:`

DESCRIPTION: Accept imaginary, fixed-point, or machine-defined literal number suffixes as GNU extensions. When this option is turned off these suffixes are treated as C++11 user-defined literal numeric suffixes. This is on by default for all pre-C++11 dialects and all GNU dialects: `'-std=c++98'`, `'-std=gnu++98'`, `'-std=gnu++11'`, `'-std=gnu++1y'`. This option is off by default for ISO C++11 onwards (`'-std=c++11'`, ...).

ALTERNATIVE: Not possible to support as it shadows C++11 and above user-defined literal numeric suffixes, source code needs to be updated.

`-Wliteral-suffix, -Wnoexcept, -Wstrict-null-sentinel, -Wno-non-template-friend, -Wno-pmf-conversions:`

DESCRIPTION: various control options over warning messages.

ALTERNATIVE: Not needed, warnings do not affect output program in any way. Clang has a different warning messaging system with different options.

## Language independent options

`-fno-diagnostics-show-caret`

DESCRIPTION: By default, each diagnostic emitted includes the original source line and a caret '^' indicating the column. This option suppresses this information.

ALTERNATIVE: `-fno-caret-diagnostics`.

## Warning options

-Waggressive-loop-optimizations, -Wclobbered, -Wconditionally-supported, -Wcoverage-mismatch, -Wno-format-contains-nul, -Wno-free-nonheap-object, -Wjump-misses-init, -Wmaybe-uninitialized, -Wunsafe-loop-optimizations, -Wlogical-op, -Wmaybe-uninitialized, -Wopenmp-simd, -Wpacked-bitfield-compat, -Wno-pedantic-ms-format, -Wno-return-local-addr, -Wstack-usage=len, -Wsuggest-attribute=pure, -Wsync-nand, -Wtrampolines, -Wunsuffixed-float-constants, -Wunused-but-set-parameter, -Wunused-but-set-variable, -Wuseless-cast, -Wvector-operation-performance

DESCRIPTION: various control options over warning messages.

ALTERNATIVE: Not needed, warnings do not affect output program in any way. Clang has a different warning messaging system with different options.

## C-only Warning Options

*-Wmissing-parameter-type*

DESCRIPTION: A function parameter is declared without a type specifier in K&R-style functions.

ALTERNATIVE: -Wunused-parameter.

*-Wtraditional, -Wtraditional-conversion, -Wold-style-declaration*

DESCRIPTION: Related to pre-standard C which is not fully supported in GCC.

ALTERNATIVE: Not needed. There's no support for pre-standard C in clang.

## Debugging options

*-fdbg-cnt-list, -fdbg-cnt=counter-value-list, -dumpspeaks, -fdump-noaddr, -fdump-unnumbered, -fdump-unnumbered-links, -fdump-translation-unit[-n], -fdump-class-hierarchy[-n], -fdump-ipa-all, -fdump-ipa-cgraph, -fdump-ipa-inline, -fdump-passes, -fdump-statistics, -fdump-tree-all, -fdump-tree-original[-n], -fdump-tree-optimized[-n], -fdump-tree-cfg, -fdump-tree-alias, -fdump-tree-ch, -fdump-tree-ssa[-n], -fdump-tree-pre[-n], -fdump-tree-ccp[-n], -fdump-tree-dce[-n], -fdump-tree-gimple[-raw], -fdump-tree-dom[-n], -fdump-tree-dse[-n], -fdump-tree-phirop[-n], -fdump-tree-phirop[-n], -fdump-tree-forwprop[-n], -fdump-tree-copyrename[-n], -fdump-tree-nrv, -fdump-tree-vect, -fdump-tree-sink, -fdump-tree-sra[-n], -fdump-tree-forwprop[-n], -fdump-tree-fre[-n], -fdump-tree-vtable-verify, -fdump-tree-vrp[-n], -fdump-final-insns=file, -fcompare-debug-second, -fcompare-debug[=opts], -fmem-report, -fmem-report-wpa, -fpre-ipa-mem-report, -fpost-ipa-mem-report, -fopt-info, -print-multi-os-directory, -print-sysroot, -gtoggle,*

DESCRIPTION: Developer options for debugging GCC.

ALTERNATIVE: Not needed. They have no meaning to the user.

*-feliminate-dwarf2-dups:*

DESCRIPTION: Compress DWARF 2 debugging information by eliminating duplicated information about each symbol. This option only makes sense when generating DWARF 2 debugging information with '-gdwarf-2'.

ALTERNATIVE: Not needed, program will still work correctly without this option. Clang\LLVM has a different compression scheme, see -gz option.

*-femit-class-debug-always:*

DESCRIPTION: Instead of emitting debugging information for a C++ class in only one object file, emit it in all object files using the class. This option should be used only with debuggers that are unable to handle the way GCC normally emits debugging information for classes because using this option increases the size of debugging

information by as much as a factor of two.

ALTERNATIVE: Not needed. This option should be used only with debuggers that are unable to handle the way GCC normally emits debugging information for classes.

**-fstack-usage:**

DESCRIPTION: Makes the compiler output stack usage information for the program, on a per-function basis. The filename for the dump is made by appending '.su' to the auxname. auxname is generated from the name of the output file, if explicitly specified and it is not an executable, otherwise it is the basename of the source file. An entry is made up of three fields:

the name of the function, a number of bytes, one or more qualifiers: static, dynamic, bounded.

ALTERNATIVE: **-fstack-size-section** in clang to emit stack size metadata and use **llvm-readelf --stack-sizes <elf file>** to emit stack usage information for the whole program.

**-fvar-tracking, -fvar-tracking-assignments:**

DESCRIPTION: Those flags affect the quality of the debug information, enabled by default on certain optimization level.

ALTERNATIVE: Not needed, clang emits high quality debug information without any extra options being required.

**-gstabs, -gstabs+**

DESCRIPTION: Produces debugging information in stabs format.

ALTERNATIVE: Not needed. Even in case of GCC RL78 we only support DWARF.

**-fno-merge-debug-strings:**

DESCRIPTION: Direct the linker to not merge together strings in the debugging information that are identical in different object files. Merging is not supported by all assemblers or linkers. Merging decreases the size of the debug information in the output file at the cost of increasing linkprocessing time. Merging is enabled by default.

ALTERNATIVE: Not needed, program will still work correctly without this option. Clang\LLVM has a different compression scheme, see **-gz** option.

**-femit-struct-debug-baseonly:**

DESCRIPTION: Emit debug information for struct-like types only when the base name of the compilation source file matches the base name of file in which the struct is defined. This option substantially reduces the size of debugging information, but at significant potential loss in type information to the debugger.

ALTERNATIVE: Not needed, reduces debug information size however it affects quality of the debug information. Clang\LLVM has a different compression scheme, see **-gz** option.

**-femit-struct-debug-reduced:**

DESCRIPTION: Emit debug information for struct-like types only when the base name of the compilation source file matches the base name of file in which the type is defined, unless the struct is a template or defined in a system header.

ALTERNATIVE: Not needed, reduces debug information size however it affects quality of the debug information. Clang\LLVM has a different compression scheme, see **-gz** option.

## Preprocessor Options

**-dN:**

DESCRIPTION: Instead of the normal output, generate a list of '#define' directives for all the predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor.

ALTERNATIVE: **-dD**.

*-imultilib dir:*

DESCRIPTION: Use dir as a subdirectory of the directory containing target-specific C++ headers.

ALTERNATIVE: Not needed. Clang\LLVM has a different multilib dir.

*-fdebug-cpp:*

DESCRIPTION: This option is only useful for debugging GCC. When used with '-E', dumps debugging information about location maps. Every token in the output is preceded by the dump of the map its location belongs to.

ALTERNATIVE: Not needed, is only useful for debugging GCC.

*-ftrack-macro-expansion:*

DESCRIPTION: Track locations of tokens across macro expansions. This allows the compiler to emit diagnostic about the current macro expansion stack when a compilation error occurs in a macro expansion. Using this option makes the preprocessor and the compiler consume more memory. The level parameter can be used to choose the level of precision of token location tracking thus decreasing the memory consumption if necessary. Value '0' de-activates this option just as if no '-ftrack-macro-expansion' was present on the command line. Value '1' tracks tokens locations in a degraded mode for the sake of minimal memory overhead. In this mode all tokens resulting from the expansion of an argument of a function-like macro have the same location. Value '2' tracks tokens locations completely. This value is the most memory hungry. When this option is given no argument, the default parameter value is '2'. Note that -ftrack-macro-expansion=2 is activated by default.

ALTERNATIVE: Not needed, clang emits diagnostic information about macro expansion stack without any extra options being required.

*-fworking-directory:*

DESCRIPTION: Enable generation of linemarkers in the preprocessor output that will let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor will emit, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC will use this directory, when it's present in the preprocessed input, as the directory emitted as the current working directory in some debugging information formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form '-fno-working-directory'. If the '-P' flag is present in the command line, this option has no effect, since no #line directives are emitted whatsoever.

ALTERNATIVE: Not needed, clang emits correct debug information in regards to the directory without any extra options being required.

*-remap:*

DESCRIPTION: Enable special code to work around file systems which only permit very short file names, such as MS-DOS.

ALTERNATIVE: Not needed, there's no support for MS-DOS or similar OS.

## Assembler options

*-a[sub-option...]*

DESCRIPTION: turn on listings

ALTERNATIVE: Not supported

*--listing-lhs-width*

DESCRIPTION: set the width in words of the output data column of the listing

ALTERNATIVE: Not supported

*--listing-lhs-width2*

DESCRIPTION: set the width in words of the continuation lines of the output data column; ignored if smaller than

the width of the first line

ALTERNATIVE: Not supported

--listing-rhs-width

DESCRIPTION: set the max width in characters of the lines from the source file

ALTERNATIVE: Not supported

--listing-cont-lines

DESCRIPTION: set the maximum number of continuation lines used for the output data column of the listing

ALTERNATIVE: Not supported

-D

DESCRIPTION: produce assembler debugging messages

ALTERNATIVE: Not supported

--debug-prefix-map OLD=NEW

DESCRIPTION: map OLD to NEW in debug information

ALTERNATIVE: -fdebug-prefix-map

--gstabs

DESCRIPTION: generate STABS debugging information

ALTERNATIVE: Not supported

--gstabs+

DESCRIPTION: generate STABS debug info with GNU extensions

ALTERNATIVE: Not supported

--gdwarf-sections

DESCRIPTION: generate per-function section names for DWARF line information

ALTERNATIVE: Not supported

-L,-keep-locals

DESCRIPTION: keep local symbols (e.g. starting with `L')

ALTERNATIVE: Not supported

--strip-local-absolute

DESCRIPTION: strip local absolute symbols

ALTERNATIVE: Not supported

--warn

DESCRIPTION: don't suppress warnings

ALTERNATIVE: Not supported

--fatal-warnings

DESCRIPTION: treat warnings as errors

ALTERNATIVE: Supported

-J

DESCRIPTION: don't warn about signed overflow

ALTERNATIVE: Not supported

-K

DESCRIPTION: warn when differences altered for long displacements

ALTERNATIVE: Not supported

-Z

DESCRIPTION: generate object file even after errors

ALTERNATIVE: Not supported

`--alternate`

DESCRIPTION: initially turn on alternate macro syntax

ALTERNATIVE: Not supported

`-f`

DESCRIPTION: skip whitespace and comment preprocessing

ALTERNATIVE: Not supported

`--hash-size=<value>`

DESCRIPTION: set the hash table size close to <value>

ALTERNATIVE: Not supported

`-M, -mri`

DESCRIPTION: assemble in MRI compatibility mode

ALTERNATIVE: Not supported

`--size-check=[error|warning]`

DESCRIPTION: ELF .size directive check (default `--size-check=error`)

ALTERNATIVE: Not supported

`--traditional-format`

DESCRIPTION: Use same format as native assembler when possible

ALTERNATIVE: Not supported

`-R`

DESCRIPTION: fold the data section into the text section

ALTERNATIVE: Not supported

`--reduce-memory-overheads`

DESCRIPTION: prefer smaller memory use at the cost of longer assembly times

ALTERNATIVE: Not supported

`--MD FILE`

DESCRIPTION: write dependency information in FILE (default none)

ALTERNATIVE: Not supported

`--defsym SYM=VAL`

DESCRIPTION: define symbol SYM to given value

ALTERNATIVE: Not supported

`--execstack`

DESCRIPTION: require executable stack for this object

ALTERNATIVE: Not supported

## Directory Options

`-ipluginidir=dir, -specs=file`

DESCRIPTION: Those options are used to modify the behavior of the compiler using plugins and are useful for GCC developers only.

ALTERNATIVE: Not needed, useful only for GCC developers.



## Linker Options

*-static-libasan, -static-libtsan, -static-liblsan, -static-libubsan, -static-libstdc++*

DESCRIPTION: Sanitizer support, not working for RL78.

ALTERNATIVE: clang has different sanitizers (see `-fsanitize=...`)

`--no-keep-memory`

DESCRIPTION: Use less memory and more disk I/O

ALTERNATIVE: Not supported

`--reduce-memory-overheads`

DESCRIPTION: Reduce memory overheads, possibly taking much longer

ALTERNATIVE: Not supported

`--relax`

DESCRIPTION: Reduce code size by using target specific optimizations

ALTERNATIVE: Not supported

`--no-relax`

DESCRIPTION: Do not use relaxation techniques to reduce code size

ALTERNATIVE: Not supported

`--strip-discarded`

DESCRIPTION: Strip symbols in discarded sections

ALTERNATIVE: Not supported

`--no-strip-discarded`

DESCRIPTION: Do not strip symbols in discarded sections

ALTERNATIVE: Not supported

`--split-by-file [=SIZE]`

DESCRIPTION: Split output sections every SIZE octets

ALTERNATIVE: Not supported

`--split-by-reloc [=COUNT]`

DESCRIPTION: Split output sections every COUNT relocations

ALTERNATIVE: Not supported

`--section-start SECTION=ADDRESS`

DESCRIPTION: Set address of named section only address

ALTERNATIVE: Supported

`-Ttext-segment ADDRESS`

DESCRIPTION: Set address of text segment

ALTERNATIVE: Not supported, use `--section-start` instead

`-Trodata-segment ADDRESS`

DESCRIPTION: Set address of rodata segment

ALTERNATIVE: Not supported, use `--section-start` instead

`-Tldata-segment ADDRESS`

DESCRIPTION: Set address of ldata segment

ALTERNATIVE: Not supported, use `--section-start` instead

`--target-help`

DESCRIPTION: Display target specific options

ALTERNATIVE: Not supported

--stats

DESCRIPTION: Print memory usage statistics

ALTERNATIVE: Silently ignored

-V

DESCRIPTION: Print version and emulation information

ALTERNATIVE: Supported

--print-output-format

DESCRIPTION: Print default output format

ALTERNATIVE: Not supported

--warn-constructors

DESCRIPTION: Warn if global constructors/destructors are seen

ALTERNATIVE: Not supported

--warn-multiple-gp

DESCRIPTION: Warn if the multiple GP values are used

ALTERNATIVE: Not supported

--warn-once

DESCRIPTION: Warn only once per undefined symbol

ALTERNATIVE: Silently ignored

--warn-orphan

DESCRIPTION: Warn if any orphan sections are encountered

ALTERNATIVE: --orphan-handling=warn

--no-warn-orphan

DESCRIPTION: Do not warn if orphan sections are encountered (default)

ALTERNATIVE: Omitted or --orphan-handling=place

--warn-section-align

DESCRIPTION: Warn if start of section changes due to alignment

ALTERNATIVE: Not supported

--warn-alternate-em

DESCRIPTION: Warn if an object has alternate ELF machine code

ALTERNATIVE: Not supported

--ignore-unresolved-symbol SYMBOL

DESCRIPTION: Unresolved SYMBOL will not cause an error or warning

ALTERNATIVE: Not supported

--no-warn-mismatch

DESCRIPTION: Don't warn about mismatched input files

ALTERNATIVE: Silently ignored

--no-warn-search-mismatch

DESCRIPTION: Don't warn on finding an incompatible library

ALTERNATIVE: Not supported

--sort-common [=ascending|descending]

DESCRIPTION: Sort common symbols by alignment [in specified order]

ALTERNATIVE: Silently ignored

--sort-section name|alignment

DESCRIPTION: Sort sections by name or maximum alignment

ALTERNATIVE: Not supported

-z common-page-size=SIZE

DESCRIPTION: Set common page size to SIZE

ALTERNATIVE: Not supported

-z defs

DESCRIPTION: Report unresolved symbols in object files.

ALTERNATIVE: Supported

-z execstack

DESCRIPTION: Mark executable as requiring executable stack

ALTERNATIVE: Supported

-z max-page-size=SIZE

DESCRIPTION: Set maximum page size to SIZE

ALTERNATIVE: Supported

-z noexecstack

DESCRIPTION: Mark executable as not requiring executable stack

ALTERNATIVE: Supported

-b TARGET, --format TARGET

DESCRIPTION: Specify the input format for following input files

ALTERNATIVE: Supported (default, elf, binary)

-R FILE, --just-symbols FILE

DESCRIPTION: Just link symbols (if directory, same as --rpath)

ALTERNATIVE: Supported

--unique [=SECTION]

DESCRIPTION: Don't merge input [SECTION | orphan] sections

ALTERNATIVE: Not supported

-Ur

DESCRIPTION: Build global constructor/destructor tables

ALTERNATIVE: Not supported

--hash-size=<NUMBER>

DESCRIPTION: Set default hash table size close to <NUMBER>

ALTERNATIVE: Not supported

--default-symver

DESCRIPTION: Create default symbol version

ALTERNATIVE: Not supported

--default-imported-symver

DESCRIPTION: Create default symbol version for imported symbols

ALTERNATIVE: Not supported

--task-link SYMBOL

DESCRIPTION: Do task level linking

ALTERNATIVE: Not supported

--version-exports-section SYMBOL

DESCRIPTION: Take export symbols list from .exports, using SYMBOL as the version.

ALTERNATIVE: Not supported

-c FILE, --mri-script FILE

DESCRIPTION: Read MRI format linker script

ALTERNATIVE: Not supported

`--default-script FILE, -dT`  
 DESCRIPTION: Read default linker script  
 ALTERNATIVE: `-T` can be used to specify a non-default linkerscript

`--force-exe-suffix`  
 DESCRIPTION: Force generation of file with `.exe` suffix  
 ALTERNATIVE: Not supported

`--traditional-format`  
 DESCRIPTION: Use same format as native linker  
 ALTERNATIVE: Not supported

`-defsym=<symbol>=<value>`  
 DESCRIPTION: Define a symbol alias  
 ALTERNATIVE: `-defsym SYMBOL=EXPRESSION`

`-fini=<symbol>`  
 DESCRIPTION: Specify a finalizer function  
 ALTERNATIVE: `--fini=SYMBOL`

`-init SYMBOL`  
 DESCRIPTION: Call SYMBOL at load-time  
 ALTERNATIVE: `--init=<symbol>` Specify an initializer function

`--no-undefined`  
 DESCRIPTION: Do not allow unresolved references in object files  
 ALTERNATIVE: `ld` has this options only for shared objects

`--demangle [=STYLE]`  
 DESCRIPTION: Demangle symbol names [using STYLE]  
 ALTERNATIVE: Supported, but STYLE can't be selected.

## Optimization options

*-faggressive-loop-optimizations, -falign-jumps, -falign-labels, -falign-loops, -fauto-inc-dec, -fbranch-probabilities, -fbranch-target-load-optimize, -fbtr-bb-exclusive, -fcaller-saves, -fcheck-data-deps, -fcombine-stack-adjustments, -fconserve-stack, -fcompare-elim, -fcprop-registers, -fcrossjumping, -fcse-follow-jumps, -fcse-skip-blocks, -fcx-fortran-rules, -fcx-limited-range, -fdce, -fdevirtualize, -fdevirtualize-speculatively, -fdse, -fearly-inlining, -fipa-sra, -fexpensive-optimizations, -ffat-lto-objects, -ffloat-store, -fexcess-precision=style, -fforward-propagate, -fgcse, -fgcse-after-reload, -fgcse-las, -fgcse-lm, -fgraphite-identity, -fgcse-sm, -fhoist-adjacent-loads, -fif-conversion, -fif-conversion2, -findirect-inlining, -finline-functions-called-once, -finline-small-functions, -fipa-cp, -fipa-cp-clone, -fipa-pta, -fipa-profile, -fipa-pure-const, -fipa-reference, -fira-algorithm=algorithm, -fira-region=region, -fira-hoist-pressure, -fira-loop-pressure, -fno-ira-share-save-slots, -fno-ira-share-spill-slots, -fisolte-erroneous-paths-dereference, -fisolte-erroneous-paths-attribute, -fivopts, -fkeep-inline-functions, -fkeep-static-consts, -flive-range-shrinkage, -floop-block, -floop-interchange, -floop-strip-mine, -floop-nest-optimize, -floop-parallelize-all, -flto-partition=alg, -flto-report, -flto-report-wpa, -fmerge-constants, -fmodulo-sched, -fmodulo-sched-allow-regmoves, -fmove-loop-invariants, -fno-branch-count-reg, -fno-defer-pop, -fno-function-cse, -fno-guess-branch-probability, -fno-peephole, -fno-peephole2, -fno-sched-interblock, -fno-sched-spec, -fno-toplevel-reorder, -fpartial-inlining, -fpeel-loops, -fpredictive-commoning, -fprofile-report, -fprofile-correction, -fprofile-values, -fprofile-reorder-functions, -free, -frename-registers, -freorder-blocks, -freorder-blocks-and-partition, -freorder-functions, -frerun-cse-after-loop, -freschedule-modulo-scheduled-loops, -frounding-math, -fsched2-use-superblocks, -fsched-pressure, -fsched-spec-load, -fsched-spec-load-dangerous, -fsched-stalled-insns-dep[=n], -fsched-stalled-insns[=n], -fsched-group-heuristic, -fsched-critical-path-heuristic, -fsched-spec-insn-heuristic, -fsched-rank-heuristic, -fsched-last-insn-heuristic, -fsched-dep-count-heuristic, -fselective-scheduling, -fselective-scheduling2, -fsel-sched-pipelining, -fsel-sched-pipelining-outer-loops, -fshrink-wrap,*

*-fsignaling-nans, -fsingle-precision-constant, -fsplit-ivs-in-unroller, -fsplit-wide-types, -fthread-jumps, -ftracer, -ftree-bit-ccp, -ftree-builtin-call-dce, -ftree-ccp, -ftree-ch, -ftree-coalesce-vars, -ftree-copy-prop, -ftree-copyrename, -ftree-dce, -ftree-dominator-opts, -ftree-dse, -ftree-forwprop, -ftree-fre, -ftree-loop-if-convert, -ftree-loop-if-convert-stores, -ftree-loop-im, -ftree-phi-prop, -ftree-loop-distribution, -ftree-loop-distribute-patterns, -ftree-loop-ivcanon, -ftree-loop-linear, -ftree-loop-optimize, -ftree-loop-vectorize, -ftree-pre, -ftree-partial-pre, -ftree-pta, -ftree-reassoc, -ftree-sink, -ftree-sra, -ftree-switch-conversion, -ftree-tail-merge, -ftree-ter, -ftree-vrp, -funroll-all-loops, -funsafe-loop-optimizations, -funswitch-loops, -fvariable-expansion-in-unroller, -fvect-cost-model, -fvpt, -fweb, -fuse-linker-plugin*

DESCRIPTION: All of the options are optimization related and there's a reasonable expectation that llvm has different optimization passes and options.

ALTERNATIVE: Not needed.

## Tool options

### llvm-objdump compatibility with rl78-elf-objdump

**-i**

DESCRIPTION: List object formats and architectures supported

ALTERNATIVE: --version

**-g**

DESCRIPTION: Display debugging information

ALTERNATIVE: --dwarf

## Assembler compatibility

The integrated assembler in clang supports the majority of directives defined by the GNU assembler. However there are a few differences which are discussed in detail below.

### .eqv symbol, expression

DESCRIPTION: The .eqv directive is like .equiv, but no attempt is made to evaluate the expression or any part of it immediately. Instead each time the resulting symbol is used in an expression, a snapshot of its current value is taken.

ALTERNATIVE: Replace with .equiv

### .fail expression

DESCRIPTION: -Generates an error or a warning. If the value of the expression is 500 or more, as will print a warning message. If the value is less than 500, as will print an error message. The message will include the value of expression. This can occasionally be useful inside complex nested macros or conditional assembly.

ALTERNATIVE: Use .warning and .error

### .hword

DESCRIPTION: This expects zero or more expressions, and emits a 16 bit number for each.

This directive is a synonym for '.short'; depending on the target architecture, it may also be a synonym for '.word'.

ALTERNATIVE: use .short

## **.lflags**

DESCRIPTION: AS accepts this directive, for compatibility with other assemblers, but ignores it.

ALTERNATIVE: not needed since it is ignored

## **.string8 "str", .string16, 32, 64**

DESCRIPTION: Copy the characters in str to the object file. You may specify more than one string to copy, separated by commas. Unless otherwise specified for a particular machine, the assembler marks the end of each string with a 0 byte. You can use any of the escape sequences described in Strings.

The variants string16, string32 and string64 differ from the string pseudo opcode in that each 8-bit character from str is copied and expanded to 16, 32 or 64 bits respectively. The expanded characters are stored in target endianness byte order.

ALTERNATIVE: .string8 same as .string and the others can be written in terms of .string

.string32 "BYE"

expands to:

.string "B\0\0\0Y\0\0\0E\0\0\0" /\* On little endian targets. \*/

.string "\0\0\0B\0\0\0Y\0\0\0E" /\* On big endian targets. \*/

## **.stabd, .stabs, .stabs**

DESCRIPTION: There are three directives that begin '.stab'. All emit symbols, for use by symbolic debuggers. The symbols are not entered in the as hash table: they cannot be referenced elsewhere in the source file.

ALTERNATIVE: Not needed. From the description .stabd, .stabs, .stabs are related to "stabs" debugging format, even in case of GCC RL78 we only support DWARF so those directives shouldn't appear in any GCC project either.

## **.mri val**

DESCRIPTION: If val is non-zero, this tells as to enter MRI mode. If val is zero, this tells as to exit MRI mode. This change affects code assembled until the next .mri directive, or until the end of the file.

ALTERNATIVE: Not needed. .mri changes to syntax to MRI mode which is specific for Motorola M68K and this has nothing to do with RL78

## **.gnu\_attribute tag,value**

DESCRIPTION: .gnu\_attribute records a GNU object attribute for this file.

ALTERNATIVE: Not needed. .gnu\_attribute is an object attribute which was developed as part of the ABI for the ARM Architecture

## **.func name[,label], .endfunc**

DESCRIPTION: .func emits debugging information to denote function name, and is ignored unless the file is assembled with debugging enabled. Only '--gstabs[+]' is currently supported. label is the entry point of the function and if omitted name prepended with the 'leading char' is used. 'leading char' is usually \_ or nothing, depending on the target. All functions are currently defined to have void return type. The function must be terminated with .endfunc.

ALTERNATIVE: Not needed. These directives are related to stabs debugging format, not relevant for us (we use DWARF)

## **.vtable\_entry table, offset**

DESCRIPTION: This directive finds or creates a symbol table and creates a VTABLE\_ENTRY relocation for it with an addend of offset.

ALTERNATIVE: Not needed. This directive is related to vtable in C++ and there is no clear example on how to do this from assembler.

## **.vtable\_inherit child, parent**

DESCRIPTION: This directive finds the symbol child and finds or creates the symbol parent and then creates a VTABLE\_INHERIT relocation for the parent whose addend is the value of the child symbol. As a special case the parent name of 0 is treated as referring to the **ABS** section.

ALTERNATIVE: Not needed. This directive is related to vtable in C++ and there is no clear example on how to do this from assembler.

## **.loc\_mark\_labels enable**

DESCRIPTION: When emitting DWARF2 line number information, the .loc\_mark\_labels directive makes the assembler emit an entry to the .debug\_line line number matrix with the basic\_block register in the state machine set whenever a code label is seen. The enable argument should be either 1 or 0, to enable or disable this function respectively.

ALTERNATIVE: Not needed. .loc\_mark\_labels is used to change the basic block flag in DWARF2, this can be used when user writes .dwarf 2 information himself which is highly unlikely. This means that the users can do this while writing the actual information without this directive.

## **Listing Directives: .list, .nolist, .eject, .psize lines , columns, .sbttl "subheading", .title "heading"**

DESCRIPTION:

.list, .nolist: Control (in conjunction with the .nolist directive) whether or not assembly listings are generated. These two directives maintain an internal counter (which is zero initially). .list increments the counter, and .nolist decrements it. Assembly listings are generated whenever the counter is greater than zero.

.eject: Force a page break at this point, when generating assembly listings.

.psize lines , columns : This directive is used to declare the number of lines and, optionally, the number of columns to use for each page, when generating listings.

.sbttl "subheading": Use subheading as the title (third line, immediately after the title line) when generating assembly listings.

.title "heading": Use heading as the title (second line, immediately after the source file name and page number) when generating assembly listings.

ALTERNATIVE: clang/llvm don't have any listing options at the moment. As the name suggest they are listing directives and do not affect the behavior/correctness of the program as such they can be safely removed.

The listings can be obtained post-build with llvm-objdump.

## C Language compatibility

Clang was intended to be compatible with GCC. In order to achieve this it has implemented the majority of GCC extensions, it even went to such extent as to define all the GCC macros, like `__GNUC__` for example, so any `#ifdef "gcc" #else "non-gcc"` will be handled in the same way as with GCC. However there are some differences: some generic (deviations from the ISO standard, like nested functions, in GCC which are not included in LLVM but those are just a few and are considered bad practice anyway). The following is the the complete list of GCC C extensions from GCC RL78 2020q2 release not supported in LLVM RL78 toolchain.

### Nested Functions

Consider the following example:

```
foo (double a, double b)
{
double square(double z) { return z*z; }
return square (a) + square (b);
}
```

It is not clear what's the origin of this extension, but the intention might be to make implement the equivalent of `private` in C. If this is the case the recommendation is to declare the function as `static` (after it has been un-nested, moved into the global scope). Like all non-ISO C extensions we don't recommend it in GCC either.

The example on the left can be modified as follows:

```
double square(double z) { return z*z; }
foo (double a, double b)
{
return square (a) + square (b);
}
```

### Constructing Calls

The GCC manual says:

"Using the built-in functions described below, you can record the arguments a function received, and call another function with the same arguments, without knowing the number or types of the arguments ...

However, these built-in functions may interact badly with some sophisticated features or other extensions of the language. It is, therefore, not recommended to use them outside very simple functions acting as mere forwarders for their arguments."

```
void * __builtin_apply_args ();
void * __builtin_apply (void (*function)(), void *arguments, size_t size);
void __builtin_return (void *result),
__builtin_va_arg_pack ();
size_t __builtin_va_arg_pack_len ();
```

Based on the description GCC manual, those builtins there have limited usage and are not recommended except for very simple cases.

There are other better, ISO C compliant, ways to write wrapper functions.



We don't recommend using this extension in GCC either.

## Pointer Arguments in Variadic Functions

Pointer arguments of any pointer type:

`va_arg (ap, void *)`

The ISO C standard defines this undefined behavior as the compatibility between the actual argument and type cannot be checked. This should be avoided.

## Attributes

Function attributes not implemented in clang Clang: `error`, `warning`, `externally_visible`, `leaf`, `no_icf`, `no_stack_limit`, `optimize`.

Label attributes not implemented in Clang: `cold`, `hot` apply only to functions.

Most of them are optimization related and there's a reasonable expectation that llvm has different optimizations:

`error/warning`: `error/warning` is returned if functions is not eliminated by optimizations; `externally_visible` disables whole program optimization on the function; `no_icf` disable icf (identical code folding) on the function; `optimize` set optimization options on the function; `hot/cold` tell the compiler if a particular path is likely/unlikely to be execute.

`no_stack_limit` -> llvm has other features to deal with the stack usage, we will investigate them as `-Wstack-usage` used in e2 studio plugins is also not available.

## Vector Extensions

Using vector instructions through built-in functions.

`__builtin_shuffle`

Clang has `__builtin_shufflevector` instead of `__builtin_shuffle`.

Please note this is not relevant for RL78 since there are no SIMD instructions in RL78.

## Miscellaneous Builtin functions

`__builtin_tgmath`, `__builtin_complex`, `__builtin_is_constant_evaluated`, `__builtin_drem`, `__builtin_dremf`, `__builtin_dreml`, `__builtin_exp10`, `__builtin_exp10f`, `__builtin_exp10l`, `__builtin_exp10l`, `__builtin_exp10l`, `__builtin_gamma`, `__builtin_fma`, `__builtin_gammaf`, `__builtin_gammal`, `__builtin_gamma_r`, `__builtin_gammaf_r`, `__builtin_gammal_r`, `__builtin_j0`, `__builtin_j0f`, `__builtin_j0l`, `__builtin_j1`, `__builtin_j1f`, `__builtin_j1l`, `__builtin_jn`, `__builtin_inf`, `__builtin_jnf`, `__builtin_jnl`, `__builtin_lgamma_r`, `__builtin_lgammaf_r`, `__builtin_lgammal_r`, `__builtin_pow10`, `__builtin_pow10f`, `__builtin_pow10l`, `__builtin_scalb`, `__builtin_fmalf`, `__builtin_scalbf`, `__builtin_scalbl`, `__builtin_significand`, `__builtin_significandf`, `__builtin_signbitf`, `__builtin_significandl`, `__builtin_sincos`, `__builtin_sincosf`, `__builtin_sincosl`, `__builtin_y0`, `__builtin_y0f`, `__builtin_y0l`, `__builtin_y1`, `__builtin_y1f`, `__builtin_y1l`, `__builtin_yn`, `__builtin_ynf`, `__builtin_ynl`.

## Bitfields

The bitfields implementation is different compared to GCC. Please see the following links for more details:

<http://lists.llvm.org/pipermail/llvm-dev/2017-October/118507.html>

<https://gcc.gnu.org/ml/gcc/2017-10/msg00192.html>  
<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1260.htm>

## Extended Asm

Clang supports GCC's extended syntax. The only difference is in what constraints are supported. Currently only two constraints are supported in Clang:

"r" for 16 bit register

"R" for 8 bit register pair.

## C++ Language compatibility

The following is the the complete list of GCC C++ extensions from GCC RL78 2020q2 release not supported in LLVM RL78 toolchain.

### Extracting the function pointer from a bound pointer to member function

Consider the following example:

```
class A {};
extern A a;
extern int (A::*fp)();
typedef int (*fp_ptr)(A *);
fp_ptr p = (fp_ptr)(a.*fp);
```

When calling non-static member functions the compiler sets the “this” pointer, does the dynamic dispatch for virtual functions etc. It looks like this extension lets all those things for the user to do, which is both difficult and error-prone.

Also the GCC documentation does not explain how such a pointer(p from the example on the left) can be used, the example on the left is all there is in the GCC documentation.

This also look like it infringes the encapsulation principle in C++ since it exposes “private” pointer from the class.

In case of LLVM the user will see a error similar to this one:

```
error: reference to non-static member function must be called; did you mean to call it with no
arguments?
fp_ptr p = (fp_ptr)(a.*fp);
^~~~~~
()
```

### Namespace Association

Consider the following example:

```
namespace std {
namespace debug {
template <class T> struct A {};
}
```

```
using namespace debug __attribute ((__strong__));
template <> struct A<int> { }; // ok to specialize

template <class T> void f (A<T>);
}

int main()
{
f (std::A<float>()); // lookup finds std::f
f (std::A<int>());
}

```

This extension has been removed from newer versions of GCC.

The GCC 4.x documentation on which RL78 is based says:

"Caution: The semantics of this extension are not fully defined. Users should refrain from using this extension as its semantics may change subtly over time. It is possible that this extension will be removed in future versions of G++."

In case of LLVM the users will see errors similar to:

```
error: use of undeclared identifier 'f'; did you mean 'std::f'?
f (std::A<float>()); // lookup finds std::f
^
std::f
:note: 'std::f' declared here
template <class T> void f (A<T>);
^

```

## ABI compatibility with GCC RL78

There's no ABI compatibility between GCC RL78 and LLVM RL78. LLVM RL78 has been implemented following the CCRL implementation not GCC RL78.

In concrete terms what this means is the following, for example, is:

-calling convention: while GCC RL78 passes all the parameters on the stack LLVM uses registers as well just like CCRL which means improved performance.

-when using `llvm-readelf` (the equivalent of `rl78-elf-readelf`) will dump information from ELF file like elf flags (-h option) and/or relocations (-r option) will only make sense on ELF files built with LLVM RL78 or CCRL not with GCC RL78. The opposite is true in case of GCC RL78 as well (dumping information with `rl78-elf-readelf` will only make sense on ELF files built with GCC RL78 not with CCRL or LLVM RL78).

As a consequence object files built with the two compilers cannot be linked together.

## Compatibility with CCRL

LLVM RL78 aims to be compatible with CCRL at both source code and binary level.

For source code compatibility as seen in chapter 2 some CCRL extensions are already available in LLVM RL78 and will aim to implement all CCRL language extensions, as defined in the CCRL user manual, in future releases. Some extensions will be available by default while some only be available using `-frenesas-extensions`.

For compatibility at binary level LLVM RL78 has followed the CCRL implementation from the beginning.

## Assembler compatibility

LLVM RL78 supports most of the CC-RL assembly expressions, operators and directives, with a couple of exceptions and restrictions.

Contrary to CC-RL, where most integers are handled as unsigned, in LLVM absolute values are handled as 64 bit signed integers.

Expressions will be evaluated at this bitwidth, then truncated if needed to the appropriate width.

This results in some differences when evaluating expressions.

For example: `-1 > 1` will evaluate to 1 (True) in CC-RL and to 0 (False) in LLVM.

While CC-RL assembler is case insensitive, currently in LLVM only the uppercase operators/directives are supported, this will be addressed in next release.

## Operators and directives

Category	Operator or Directive	Status	Notes
Arithmetic operators	+, -, *, /, %, +sign, -sign	Supported	
Bit logic operators	~, &,  , ^	Supported	
Relational operators	"==, !=, >, >=, <, <="	Supported	
Logical operators	&&,	Supported	
Shift operators	>>, <<	Supported	
Byte separation operators	HIGH	Supported	
	LOW	Supported	
2-byte separation operators	HIGHW	Supported	
	LOWW	Supported	
	MIRHW	Supported	
	MIRLW	Supported	
	SMRLW	Supported	

Special operators	DATAPOS	Supported*	
	BITPOS	Supported*	
Section operators	STARTOF	Supported	
	SIZEOF	Supported	
Other operator	( )	Supported	
Bit position specifier		Supported	
Section definition directives	.SECTION,	Supported*, with exceptions	Unsupported relocation attributes: SBSS_BIT, BSS_BIT, BIT_AT
	.CSEG	Supported*	
	.DSEG	Supported*	
	.BSEG	Not supported	
	.ORG	Supported*	
	.OFFSET	Supported*	
Symbol definition directives	.EQU	Supported*	
	.SET	Supported*	
Data definition/Area reservation directives	.DB	Supported*	
	.DB2	Supported*	
	.DB4	Supported*	
	.DB8	Supported*	
	.DS	Supported*	

	.DBIT	Not supported	
	.ALIGN	Supported <sup>*</sup> , with restrictions	Can be aligned only to powers of two.
External definition/ External reference directives	.PUBLIC	Supported <sup>*</sup>	
	.EXTERN	Supported <sup>*</sup>	
	.EXTBIT	Not supported	
Compiler output directives	.LINE	Parsed <sup>*</sup> , but ignored	
	.STACK	Parsed <sup>*</sup> , but ignored	
	._LINE_TOP	Parsed <sup>*</sup> , but ignored	
	._LINE_END	Parsed <sup>*</sup> , but ignored	
	.TYPE	Parsed <sup>*</sup> , but ignored	
	.VECTOR	Parsed <sup>*</sup> , but ignored	
Macro directives	.MACRO	Supported <sup>*</sup>	
	.LOCAL	Supported <sup>*</sup>	
	.REPT	Supported <sup>*</sup>	
	.IRP	Supported <sup>*</sup>	
	.EXITM	Supported <sup>*</sup> , with exceptions	Mixing them with conditional assembly control instructions is not supported.

	.EXITMA	Supported <sup>*</sup> , with exceptions	Mixing them with conditional assembly control instructions is not supported.
	.ENDM	Supported <sup>*</sup>	
Branch directives	.Bcond	Not supported	
File input control instructions	INCLUDE	Supported <sup>*</sup>	
	BINCLUDE	Supported <sup>*</sup>	
Mirror source area reference control instructions	MIRROR	Not supported	
Assembler control instructions	NOWARNING, WARNING	Supported <sup>*</sup>	
Conditional assembly control instructions	IFDEF, IFNDEF, IF, IFN, ELSEIF, ELSEIFN, ELSE, ENDIF	Supported <sup>*</sup> , with exceptions	Mixing them with macro directives are not supported.

<sup>\*</sup> Only when -frenesas-extensions is specified.

## Assembler generated symbols

### @\$IMM\_<constant value>

Currently, the compiler will create local imm symbols for all unique constants that appear in an expression. This will be improved in a future release.

### .\$\$\$<symbol-name>

When defining a bit positional symbol, for example:

```
SYM1 .EQU 0x1234.2
```

The compiler will create two symbols (SYM1 and .\$\$\$SYM1), to store the address and the bit position respectively.

## ABI compatibility with CC-RL

LLVM RL78 has followed the CC-RL specification for compatibility at binary level which allows linking together object files produced by both compilers with the following restrictions:

- The comparability only covers the C language at the moment.
- The library format(.lib in CC-RL, .a in case of LLVM) are different, as a consequence LLVM linker can't read a CCRL .lib file while CCRL linker can't read an LLVM .a file.

-Some library functions which have pointers as parameters (printf, scanf, vscanf, etc) have different declarations (when the first parameter is a near pointer in case of LLVM and a far pointer in case of CC-RL) as a consequence there can't be no mixing using the declaration (from .h) from one toolchain and the definition from the other toolchain.

-Both the C and runtime libraries are larger in LLVM than CC-RL. When using functions not present in the CC-RL library this will lead to link errors when linking with CC-RL. In such cases it is recommended to link using LLVM.

-Any specific metadata present in the object files generated by one compiler is unrecognised by the other toolchain linker. This mainly includes link time optimisations (-Owhole-program in case of CC-RL and -flto in case of LLVM) This is why it is recommended not to use link time optimisations when linking object files produced by the other compiler. It also recommended to use -fno-addrsig when compiling an object file with LLVM which will be linked using CC-RL.

## Libgen

The GNU library generator tool `libgen`, builds the Newlib, Newlib\_nano, Compilerrt, Libcxx, and Libcxxabi libraries with the user-specified options for the Renesas RL78 target. It helps in generating the libraries that are fine-tuned with the user's application.

The `libgen` operates in two modes viz. command-line mode and interactive mode.

In the command line mode, the entire command line consisting of options and their suboptions has to be provided.

In the interactive mode, a menu is displayed on the command line. Following are the various menu options,

### **Select Library**

Supported options are newlib, newlib\_nano, compilerrt, libcxx and libcxxabi. By default 'newlib' will be selected.

### **Enter Compiler Options**

Specify compiler options. Any kind of compiler option can be specified, from optimization options to defines which can change the configuration of the libraries being built, for example passing -D\_WANT\_IO\_C99\_FORMATS (see newlib configure file) when building newlib enables extra printf format specifiers (also grows the code size of newlib).

### **Enter Assembler Options**

Specify assembler options.

### **Enter Path & Name for Library**

Specify the path and name of the archive. By default library name is 'libout.a'.

The `libgen` utility builds the user-selected library from the library sources installed with the toolchain. The library sources will be compiled using the clang utility and archived using the llvm-`ar` utility.

The options which can be used with `libgen` are listed below.

### **-I,--interactive**

Specify the command line options in user-friendly interactive mode.

### **-S,--select-lib=**

Specify the sources to be used for building the library. Supported options are newlib, newlib\_nano, compilerrt, libcxx and libcxxabi. If nothing is specified by the user under this option, 'newlib' will be selected.



**-C,--compiler-options=**

Specify the compiler options i.e. cpu specific options like -mcpu=S3 and/or optimization options like -O2, -Os, etc. The sub-options should be separated by ','.

**-A,--assembler-options=**

Specify the assembler options (in case of assembly files). These options should not be prefixed with '-Wa,'. The sub-options should be separated by ','.

**.-o,--output=**

Specify the name for the library archive. By default, the library is generated in the current working directory for command-line applications.

**-h,--help**

Print (on the standard output) the description of the command line options supported by the 'libgen' tool.

**-v,--version**

Show the version number of the libgen and exit.