Optimum Code Generation for 251 Microcontroller Applications with more than 64Kbytes code size.   *Application Note 116*

November 12, 1997, Munich, Germany

by   Reinhard Keil, Keil Elektronik GmbH rk@keil.com ++49 89 456040-13

You can generate applications with more than 64KB code size using a 251 CPU rather than an 8051. Most 251 CPU's support 256Kbytes address space which can be used for program code or variables. In contrast to the 8051 CPU where large programs can only be done with code banking, the 251 CPU supports the ECALL and ERET instructions which use 24-bit addresses. The ECALL instruction is one byte longer and uses one byte more stack space compared to LCALL. Therefore it is more optimal to use LCALL whenever possible. This application note shows you how to improve code density on big applications using the 251 microcontroller and the Keil C251 C Compiler. Source code for this application note is located in file named 116.zip on this CD.

The C251 Version 2 compiler controls the use of CALL & RET with the ROM compiler directive as listed in the following table. For applications with program code requirements > 64KB you must use the ROM(HUGE) directive. It should be noted that ROM(HUGE) does not affect any variable placements.

Memory Size	Description
ROM (SMALL)	**CALL** and **JMP** instructions are coded as **ACALL** and **AJMP**. The maximum program size may be 2 Kbytes. The entire program must be allocated within the 2 Kbyte program memory space.
ROM (COMPACT)	**CALL** instructions are coded as **LCALL**. **JMP** instructions are coded as **AJMP** within a function. The size of a function must not exceed 2 Kbytes. The entire program may, however, comprise a maximum of 64 Kbytes.
ROM (LARGE)	**CALL** and **JMP** instructions are coded as **LCALL** and **LJMP**. This allows you to use the entire address space without any restrictions. Program size is limited to 64 Kbytes. Function size is also limited to 64 Kbytes.
ROM (MEDIUM)	External targets are coded as **LCALL**, targets within the current code module are coded as **AJMP** or **ACALL**. A single code segment is limited to 2 Kbytes of code.
ROM (HUGE)	All function targets not explicitly modified with **near** are invoked with **ECALL** instructions. Intra-segment branches are codes with short jumps or **LJMP**'s, depending on the target distance.

When you apply ROM(HUGE), the C251 compiler uses ECALL & ERET for instruction for function calls. However, C251 provides you several ways to optimize the program:

- you can use the near keyword to generate code with LCALL & RET instruction.

- when a function is static and no address (function pointer) is taken from that function, C251 automatically defines that function as near and generates LCALL & RET to that function. Therefore it is a good practice to declare functions as static, when they are not called from other modules.

The following code examples show you the optimization potential of C251.

```
   1
   2              // C251 automatically inserts LCALL's to static functions
   3              static void near_func (void)  {
   4   1              ;
   5   1          }
   6
   7
   8              // if a function is public (without the C static attribut
   9              // a ECALL and ERET is required in ROM(HUGE) applications
  10              void far_func (void)  {
  11   1              ;
  12   1          }
  13
  14
  15              void main (void)  {
  16   1            near_func ();
  17   1            far_func ();
  18   1          }
```

```
ASSEMBLY LISTING OF GENERATED OBJECT CODE
;       FUNCTION near_func (BEGIN)
000000 22              RET
;       FUNCTION near_func (END)


;       FUNCTION far_func? (BEGIN)
000001 AA              ERET
;       FUNCTION far_func? (END)


;       FUNCTION main? (BEGIN)
000002 120000     R   LCALL     near_func
000005 9A000000   R   ECALL     far_func?
000009 AA              ERET
;       FUNCTION main? (END)
```

EXAMPLE C SOURCE CODE

Sometimes it is desired to have LCALL's even between modules. In this way you can optimize the program code size and execution speed of an application. The problem here is that you must ensure that all modules are in the same physical 64KB segment. With Keil C251 this can be easily done with the #pragma USERCLASS (UCODE = *name*) directive.

If you want to move several modules into the same 64KB segment, just use the #pragma USERCLASS (UCODE = BANK0) directive. Then the CLASS name of all segments set to ECODE_BANK0. You can then locate this class into one physical 64KB segment with the L251 CLASSES directive.

This ensures that it is always possible to use the *near* function prefix for function calls between these modules. Don't worry if you make any mistakes, the L251 linker will generate a FIXUP error, if you it is not possible to reach a function with a LCALL instruction.

The following program example details this programming technique.

```
#include <reg251s.h>              /* special function register declarations  */
                                  /* for the intended 80251 derivative       */

#include <stdio.h>                /* prototype declarations for I/O functions */

extern void bank1 (void);         /* function will be called with ECALL       */

static void near_func (void)  {   /* C251 will generate LCALL to static funcs */
  printf ("This is a near function in module main\n");
}

/****************/
/* main program */
/****************/
void main (void)  {               /* execution starts here after stack init   */
#ifndef MCB251
  SCON  = 0x50;                   /* SCON: mode 1, 8-bit UART, enable rcvr     */
  TMOD |= 0x20;                   /* TMOD: timer 1, mode 2, 8-bit reload       */
  TH1   = 0xf3;                   /* TH1:  reload value for 2400 baud          */
  TR1   = 1;                      /* TR1:  timer 1 run                         */
  TI    = 1;                      /* TI:   set TI to send first char of UART   */
#endif

  printf ("This is the main function\n");
  near_func ();                   /* LCALL in same module to static function   */
  bank1 ();                       /* ECALL to function in different module     */
  printf ("Bank in main function\n");
}
```

Module: MAIN.C (code goes into class ECODE)

```
#pragma USERCLASS (UCODE = BANK1)

#include <stdio.h>

extern void near func1bank1 (void);  // LCALL generated

void bank1 (void)  {    // function will be called with ECALL
  printf ("This is bank 1\n");
  func1bank1 ();
}
```

Module: BANK1.C (code goes into class ECODE_BANK1)
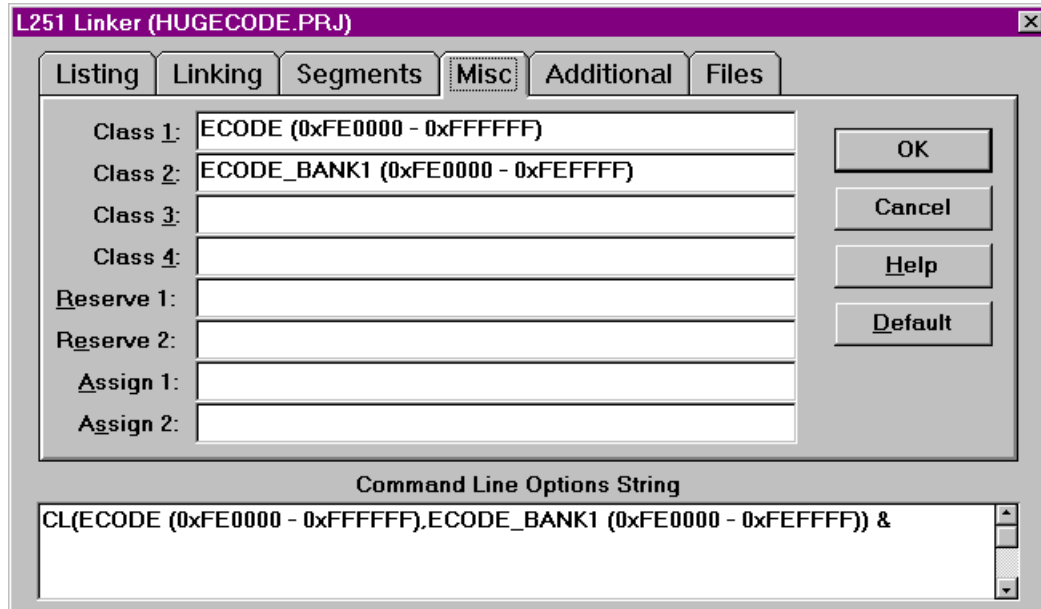
```
#pragma USERCLASS (UCODE = BANK1)

#include <stdio.h>

void near func1bank1 (void)  {  // function will be called with LCALL
  printf ("This is function 1 in bank1\n");
}
```

Module: FUNC1B1.C (code goes into class ECODE_BANK1)

To ensures that the memory class ECODE_BANK1 is locate in a 64KB bank, just enter the following CLASSES directive in µVision.

L251 Linker (HUGECODE.PRJ)

| Listing | Linking | Segments | Misc | Additional | Files |

Class 1: `ECODE (0xFE0000 - 0xFFFFFF)`

Class 2: `ECODE_BANK1 (0xFE0000 - 0xFEFFFF)`

Class 3:

Class 4:

Reserve 1:

Reserve 2:

Assign 1:

Assign 2:

OK

Cancel

Help

Default

Command Line Options String

`CL(ECODE (0xFE0000 - 0xFFFFFF),ECODE_BANK1 (0xFE0000 - 0xFEFFFF)) &`

µVision L251 Classes Setup

16990 Dallas Parkway • Suite 120 • Dallas, Texas • 75248 • 800-348-8051