Optimization in the context of the GNU Compiler Collection (GCC)

Optimization in the context of the GNU Compiler Collection (GCC) involves adjusting the code compilation process to improve various aspects of the generated executable, such as speed, size, and efficiency. GCC offers several optimization levels that control the degree and type of optimizations applied.

Optimization Levels in GCC

-O0: No optimization (default)

- This level disables all optimization techniques. The primary focus is on reducing the compilation time and improving the debugging experience. It preserves the original code structure as much as possible, which helps with debugging.

-O1: Basic optimization

- This level enables simple optimizations that do not significantly increase the compilation time. These optimizations improve the performance of the generated code without greatly affecting its size. Examples include removing redundant instructions and simplifying control flows.

-O2: Further optimization

- This level includes all -O1 optimizations and adds more aggressive techniques that can significantly improve the performance of the generated code. It focuses on reducing code size and execution time while ensuring that the compilation process remains reasonably fast. Common optimizations at this level include inlining of functions, vectorization, and loop unrolling.

-O3: Maximum optimization

- This level includes all -O2 optimizations and enables even more aggressive techniques that can further enhance performance. However, it may increase the size of the generated code and the compilation time. Examples of additional optimizations include aggressive function inlining and better use of vector instructions.

-Os: Optimize for size

- This level aims to reduce the size of the generated code while applying optimizations that do not significantly increase the code size. It is similar to -O2 but with a focus on minimizing the code footprint, making it ideal for embedded systems with limited memory.

-Ofast: Fastest possible code

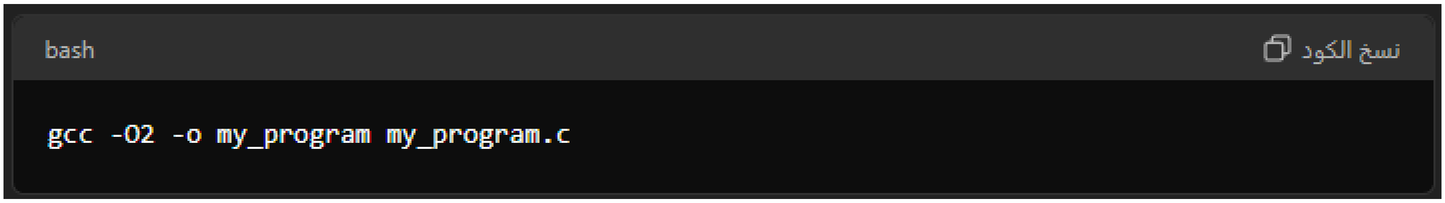
- This level includes all -O3 optimizations and applies additional aggressive techniques that may not strictly adhere to language standards. It aims to generate the fastest possible code but can result in code that is less portable or less predictable.

-Og: Optimization for debugging

- This level is designed to offer a good balance between optimization and debugging. It enables optimizations that do not interfere with the debugging experience, making it easier to debug optimized code.

Usage

To use these optimization levels, you can pass the appropriate flag to GCC during compilation. For example:



```bash
gcc -O2 -o my_program my_program.c
```

This command tells GCC to compile my_program.c with the -O2 optimization level, generating an executable named my_program.

Choosing the Right Level

-O0: Use during development and debugging.

-O1: Use when you want some performance improvements without significantly increasing the compilation time.

-O2: A good balance between performance and compilation time for most production code.

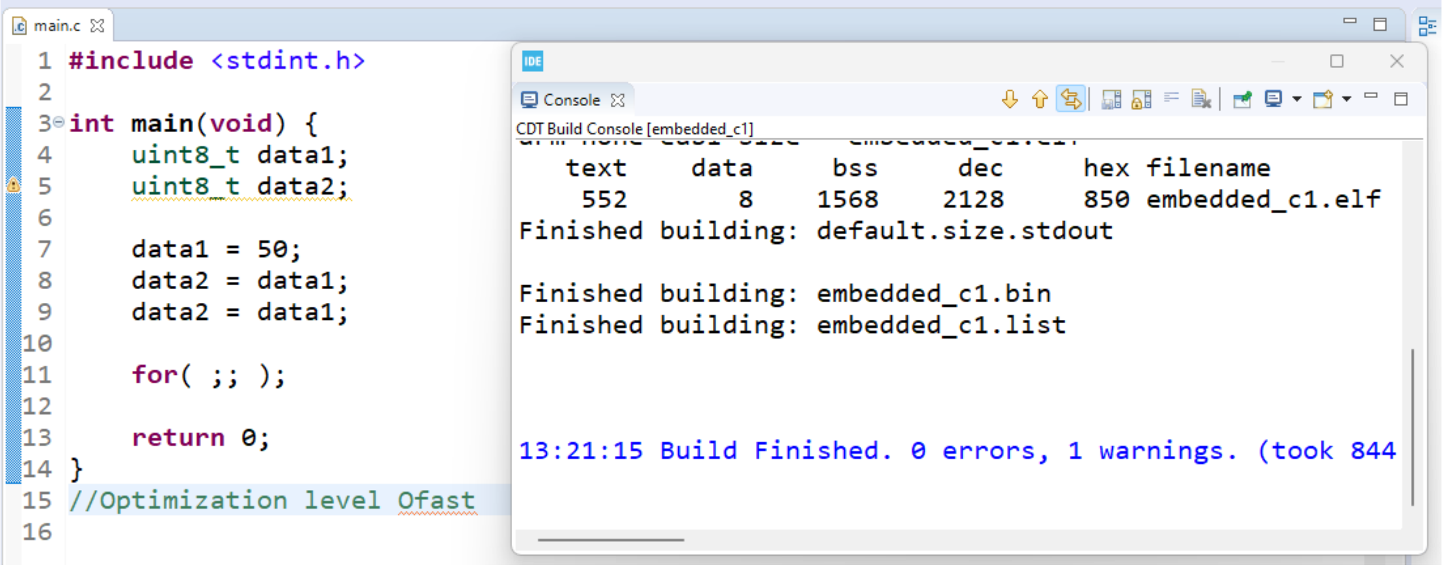
-O3: Use for compute-intensive applications where performance is critical.

-Os: Ideal for embedded systems or applications where memory is constrained.

-Ofast: Use when maximum performance is needed and strict adherence to standards is not a concern.

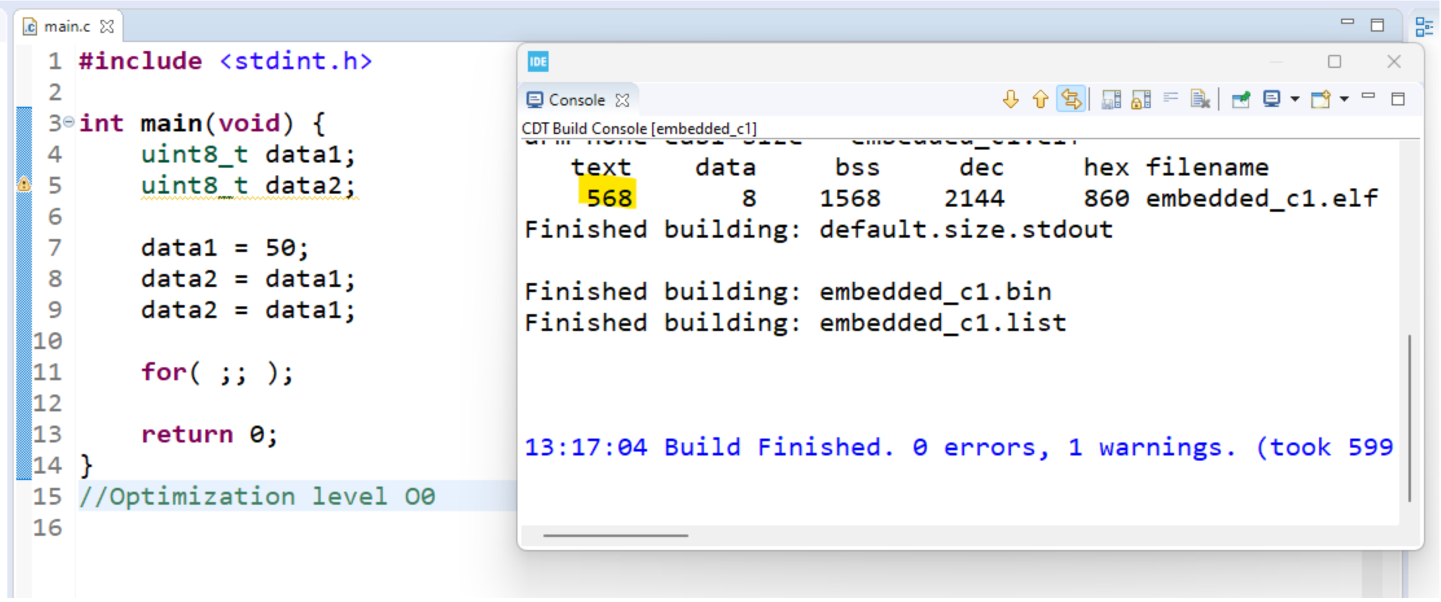
-Og: Use during development when you want some optimizations without sacrificing debugging capabilities.

if the code crashes and the program doesn't end or gives unexpected results. In order to understand where is the problem it's necessary to open the Disassembly section in the Debugging mode and control in the register windows how the data is trasferred to registers



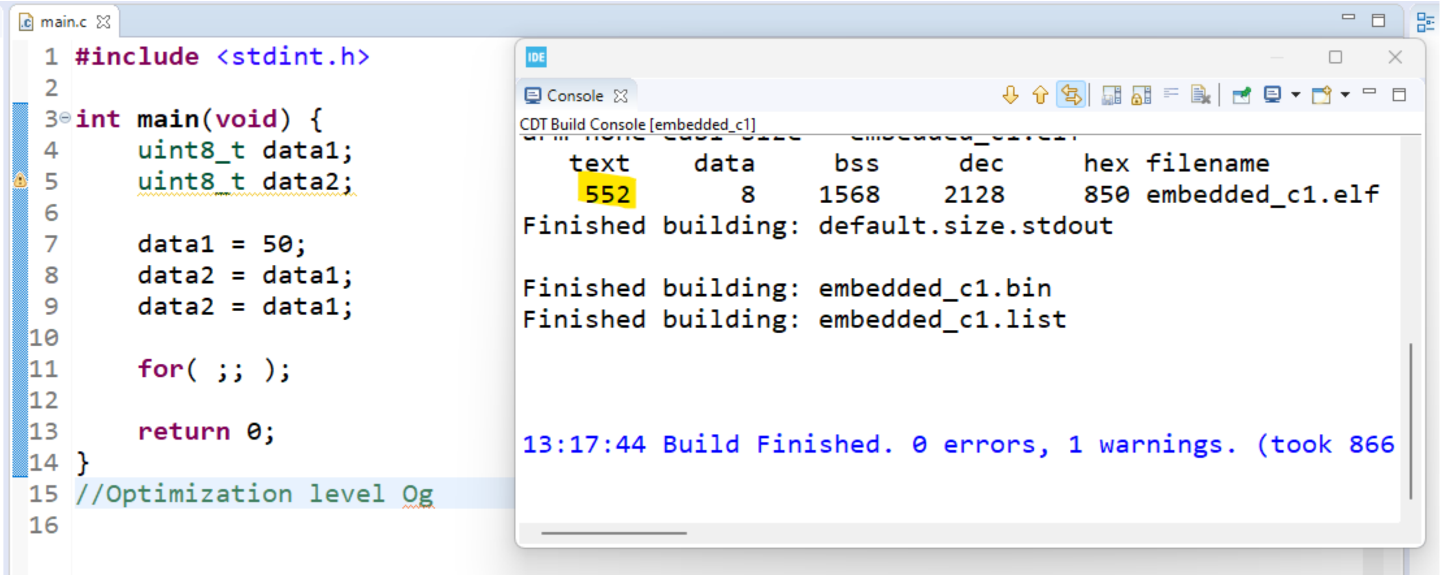
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level Ofast
```

```
arm-none-eabi-size   embedded_c1.elf
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:21:15 Build Finished. 0 errors, 1 warnings. (took 844
```



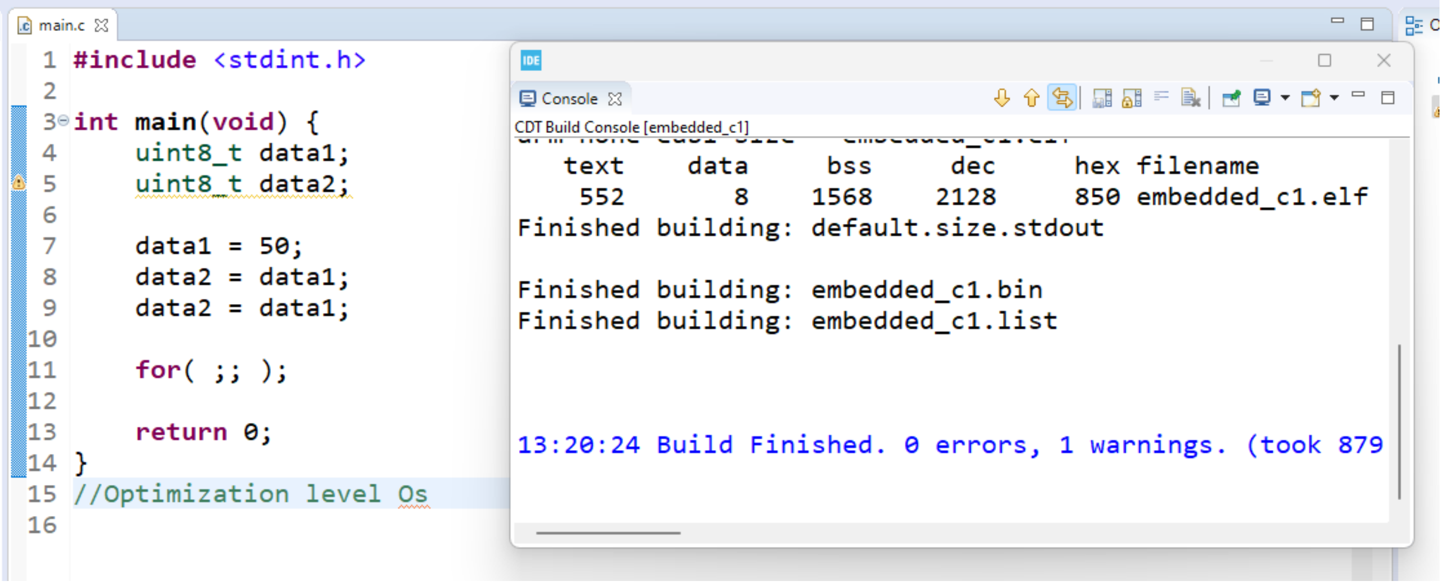
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level O0
```

```
   text    data     bss     dec     hex filename
    568       8    1568    2144     860 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:17:04 Build Finished. 0 errors, 1 warnings. (took 599
```



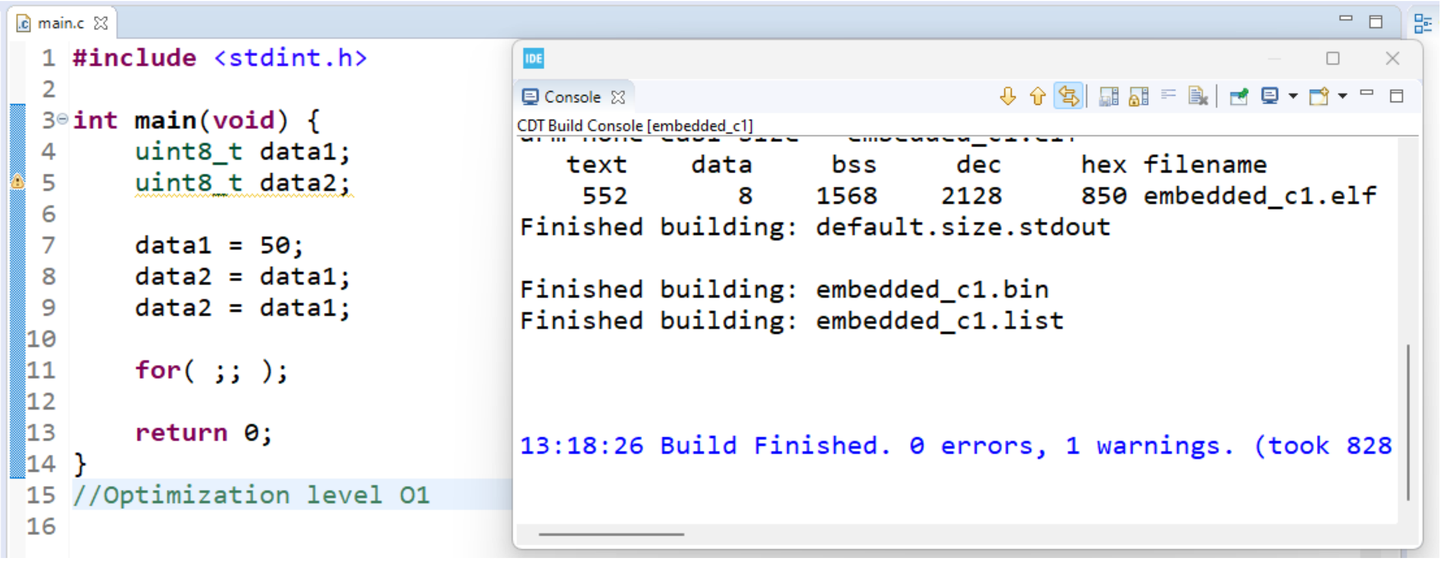
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level Og
```

```
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:17:44 Build Finished. 0 errors, 1 warnings. (took 866
```



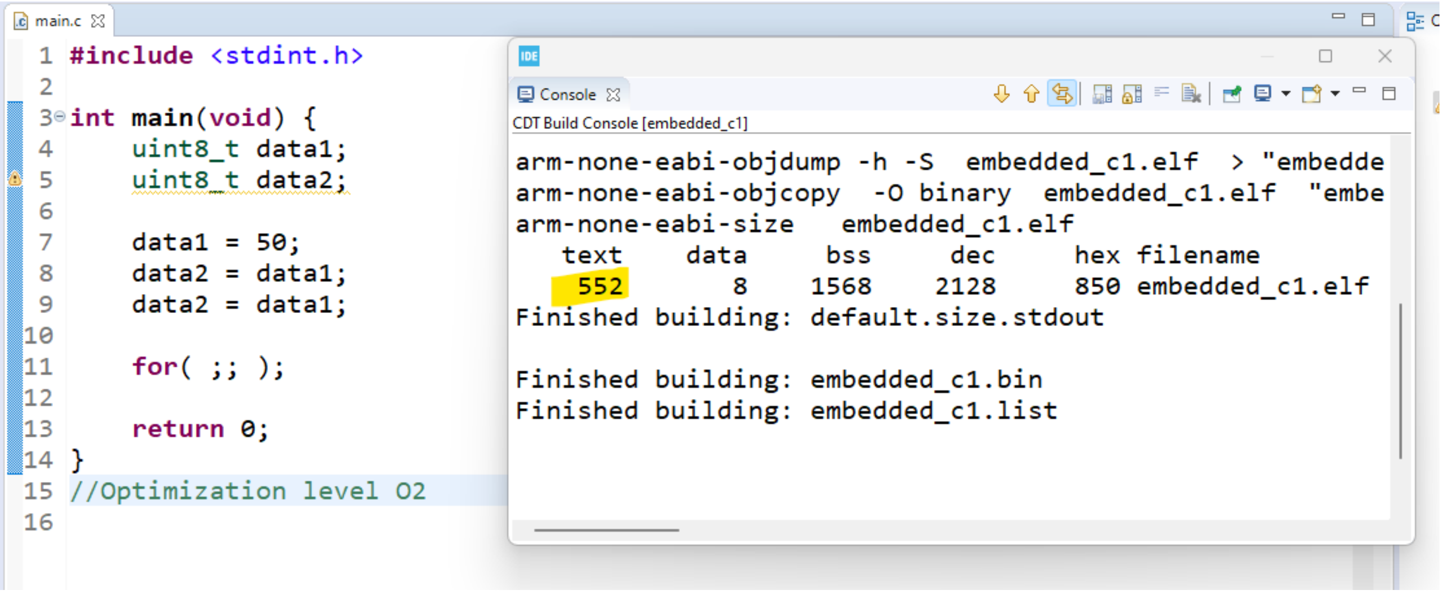
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level Os
```

```
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:20:24 Build Finished. 0 errors, 1 warnings. (took 879
```



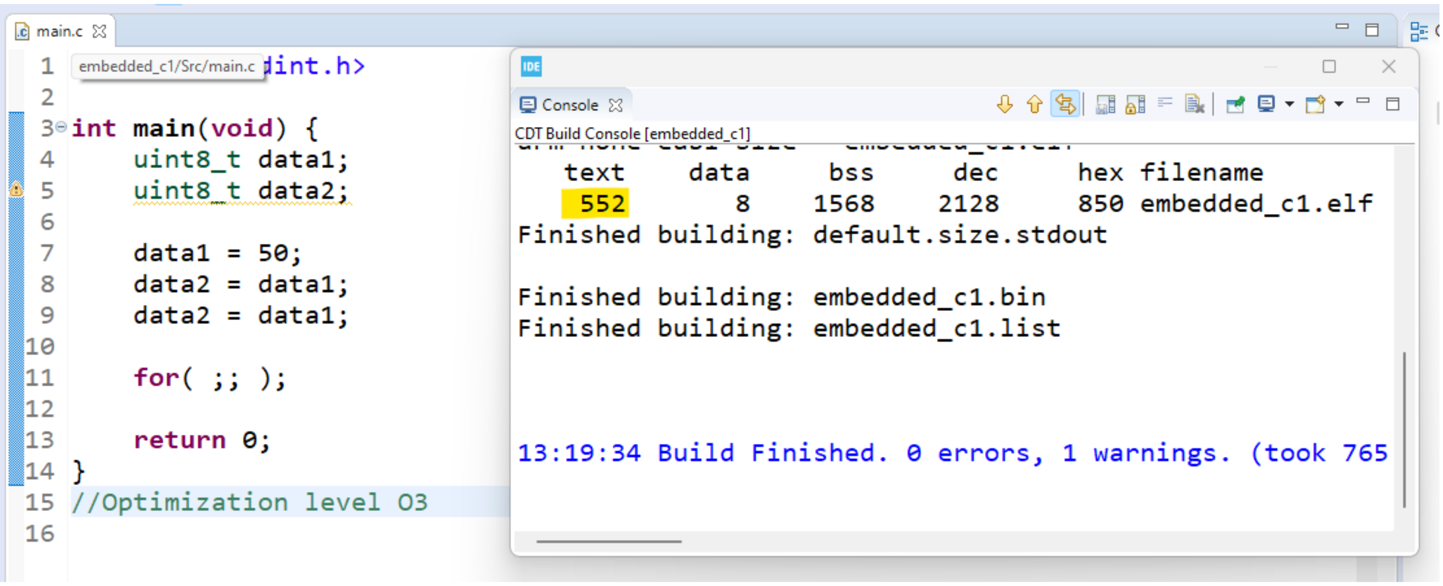
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level O1
```

```
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:18:26 Build Finished. 0 errors, 1 warnings. (took 828
```



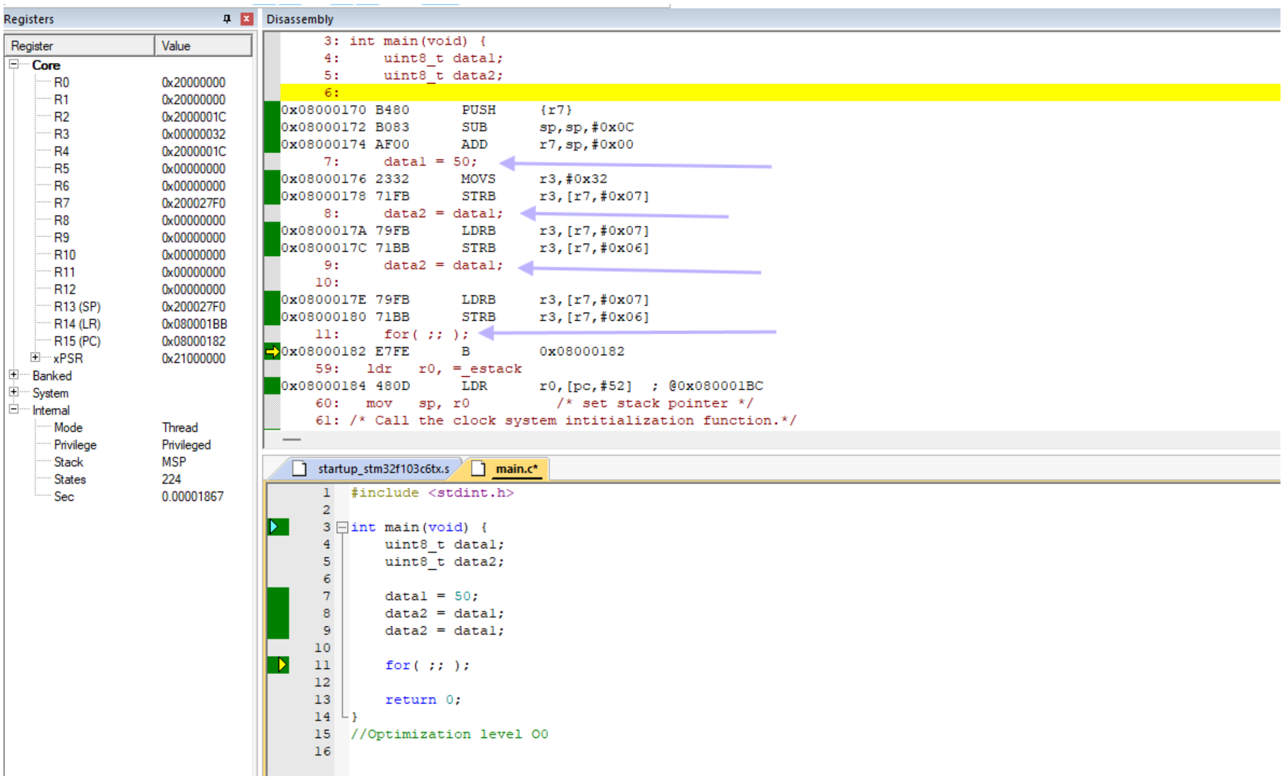
```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level O2
```

```
arm-none-eabi-objdump -h -S  embedded_c1.elf  > "embedde
arm-none-eabi-objcopy  -O binary  embedded_c1.elf  "embe
arm-none-eabi-size   embedded_c1.elf
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list
```



```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level O3
```

```
   text    data     bss     dec     hex filename
    552       8    1568    2128     850 embedded_c1.elf
Finished building: default.size.stdout

Finished building: embedded_c1.bin
Finished building: embedded_c1.list



13:19:34 Build Finished. 0 errors, 1 warnings. (took 765
```



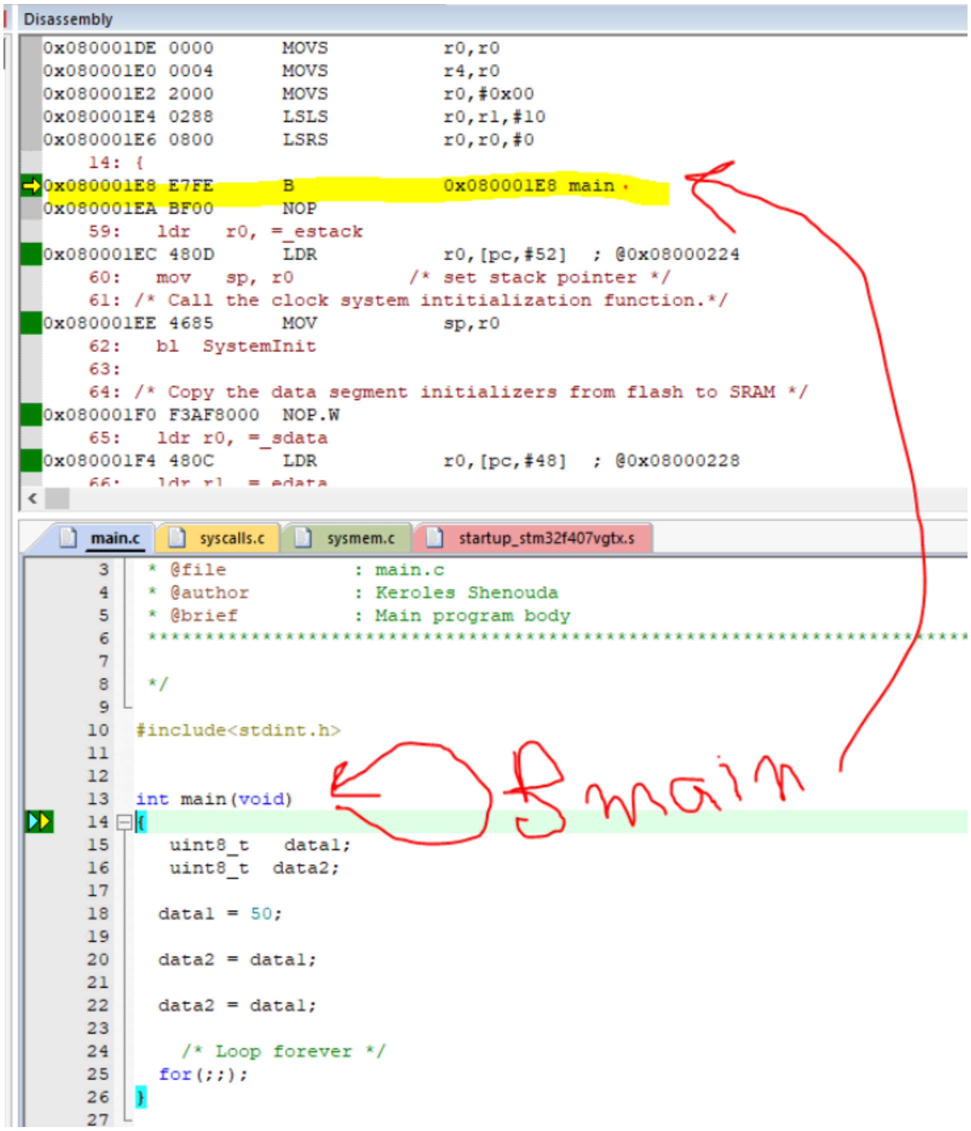
Registers

Register	Value
Core	
R0	0x20000000
R1	0x20000000
R2	0x2000001C
R3	0x00000032
R4	0x2000001C
R5	0x00000000
R6	0x00000000
R7	0x200027F0
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200027F0
R14 (LR)	0x080001BB
R15 (PC)	0x08000182
xPSR	0x21000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	224
Sec	0.00001867

Disassembly

```
     3: int main(void) {
     4:     uint8_t data1;
     5:     uint8_t data2;
     6:
0x08000170 B480      PUSH     {r7}
0x08000172 B083      SUB      sp,sp,#0x0C
0x08000174 AF00      ADD      r7,sp,#0x00
     7:     data1 = 50;
0x08000176 2332      MOVS     r3,#0x32
0x08000178 71FB      STRB     r3,[r7,#0x07]
     8:     data2 = data1;
0x0800017A 79FB      LDRB     r3,[r7,#0x07]
0x0800017C 71BB      STRB     r3,[r7,#0x06]
     9:     data2 = data1;
    10:
0x0800017E 79FB      LDRB     r3,[r7,#0x07]
0x08000180 71BB      STRB     r3,[r7,#0x06]
    11:     for( ;; );
0x08000182 E7FE      B        0x08000182
    59:   ldr   r0, =_estack
0x08000184 480D      LDR      r0,[pc,#52]  ; @0x080001BC
    60:   mov   sp, r0          /* set stack pointer */
    61: /* Call the clock system intitialization function.*/
```

```c
#include <stdint.h>

int main(void) {
    uint8_t data1;
    uint8_t data2;

    data1 = 50;
    data2 = data1;
    data2 = data1;

    for( ;; );

    return 0;
}
//Optimization level O0
```



Disassembly

```
0x080001DE 0000      MOVS        r0,r0
0x080001E0 0004      MOVS        r4,r0
0x080001E2 2000      MOVS        r0,#0x00
0x080001E4 0288      LSLS        r0,r1,#10
0x080001E6 0800      LSRS        r0,r0,#0
    14: {
0x080001E8 E7FE      B           0x080001E8 main
0x080001EA BF00      NOP
    59:   ldr   r0, =_estack
0x080001EC 480D      LDR         r0,[pc,#52]  ; @0x08000224
    60:   mov   sp, r0          /* set stack pointer */
    61: /* Call the clock system intitialization function.*/
0x080001EE 4685      MOV         sp,r0
    62:   bl  SystemInit
    63:
    64: /* Copy the data segment initializers from flash to SRAM */
0x080001F0 F3AF8000  NOP.W
    65:   ldr r0, =_sdata
0x080001F4 480C      LDR         r0,[pc,#48]  ; @0x08000228
    66:   ldr r1, =_edata
```

```c
 * @file           : main.c
 * @author         : Keroles Shenouda
 * @brief          : Main program body
 ******************************************************************************

 */

#include<stdint.h>


int main(void)
{
   uint8_t   data1;
   uint8_t  data2;

  data1 = 50;

  data2 = data1;

  data2 = data1;

    /* Loop forever */
  for(;;);
}
```