

Zad 5.

piątek, 24 marca 2023 17:09

Zadanie 5. Zaimplementuj w asemblerze x86-64 procedurę konwertującą liczbę typu «uint32_t» między formatem *little-endian* i *big-endian*. Argument funkcji jest przekazany w rejestrze %edi, a wynik zwracany w rejestrze %eax. Należy użyć instrukcji cyklicznego przesunięcia bitowego «ror» lub «rol».

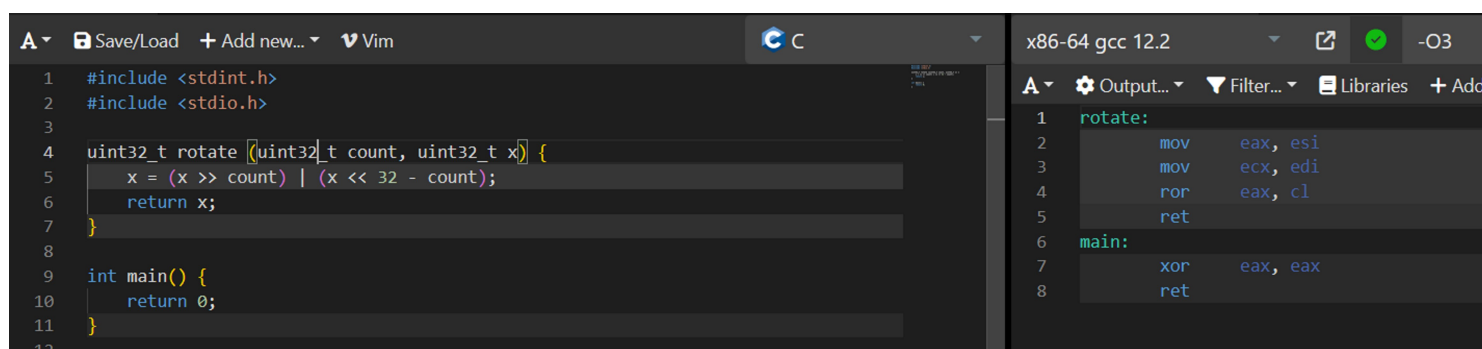
Podaj wyrażenie w języku C, które kompilator optymalizujący przetłumaczy do instrukcji «ror» lub «rol».

```
1  convert: movl %edi, %eax # %eax = A B C D
2           rorw $8, %ax  # %eax = A B D C
3           rorl $16, %eax # %eax = D C A B
4           rorw $8, %ax  # %eax = D C B A
5           ret
```

.asm

```
11 // ror $count, %edi
12 x = (x >> count) | (x << 32 - count);
```

.C



The screenshot shows a code editor with two panes. The left pane displays C code for a rotate function and its main test function. The right pane shows the corresponding assembly code generated by gcc 12.2 for x86-64 architecture, with optimization level -O3. The assembly code defines a 'rotate' function that takes 'esi' and 'edi' registers as arguments and returns the result in 'eax'. The 'main' function calls 'rotate' and returns 0.

```
1  #include <stdint.h>
2  #include <stdio.h>
3
4  uint32_t rotate(uint32_t count, uint32_t x) {
5      x = (x >> count) | (x << 32 - count);
6      return x;
7  }
8
9  int main() {
10     return 0;
11 }
12
```

x86-64 gcc 12.2 -O3

```
1  rotate:
2      mov     eax, esi
3      mov     ecx, edi
4      ror     eax, cl
5      ret
6
7  main:
8      xor     eax, eax
9      ret
```