void Exercicio1(void) {

int lista[] = { 1,2,3,4,2,1 };

int valor = 1;

int cont = 0;

\_\_asm {

mov ecx, 0

START:

mov eax, DWORD PTR[lista + ecx \* 4]

cmp eax, valor

je FIND

inc ecx

cmp ecx, 6

jg END

jmp START

FIND :

inc cont

inc ecx

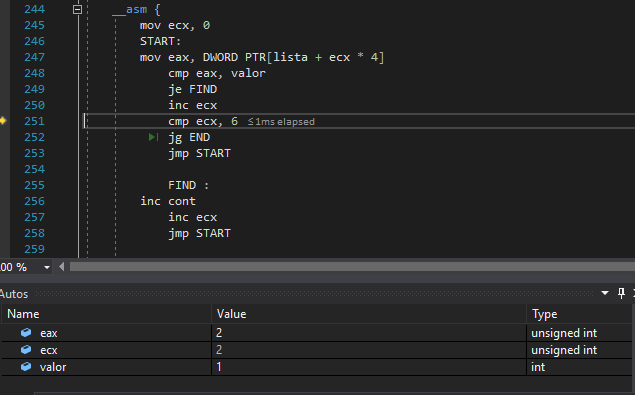
jmp START

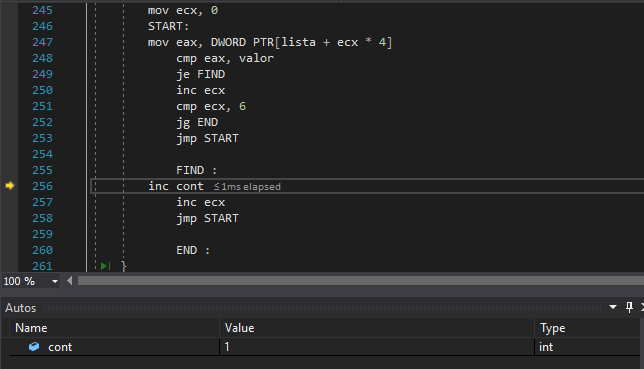
END :

}

}

// valor aparece x vezes





Cada vez que acha o valor requerido, conta +1 na variável “cont”.

void Exercicio2(void) {

char string1[] = "teste\0";

char string2[] = "abcde\0";

bool result = true;

\_\_asm {

mov ecx, 0

START:

mov eax, DWORD PTR[string1 + ecx]

cmp eax, DWORD PTR[string2 + ecx]

je FIND

mov result, 0

jmp END

FIND:

inc ecx

cmp ecx, 6

jg END

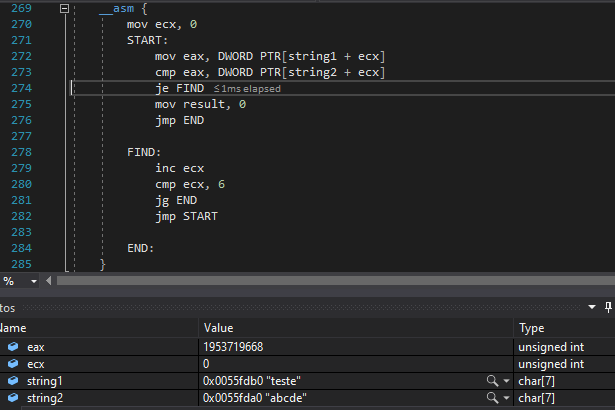
jmp START

END:

}

}

// teste string



Compara cada posição da “string1” com a “string2”, até o tamanho da string, se não houver algum igual, é falso.

void Exercicio3(void) {

int lista[] = { 20,15,10,18,5,7 };

\_\_asm {

}

}

// ordem crescente ----- INCOMPLETO

Exercício de número 3 não completo pela falta de conhecimento na hora de retirar o valor da lista.

void Exercicio4(void) {

int lista[] = { 5, 10, 15, 20, 25 };

int soma = 0, result, cinco = 5;

\_\_asm {

mov ecx, 0

START:

mov eax, DWORD PTR[lista + ecx\*4]

add eax, [soma]

mov [soma], eax

inc ecx

cmp ecx, 5

jl START

mov edx, 0

mov eax, [soma]

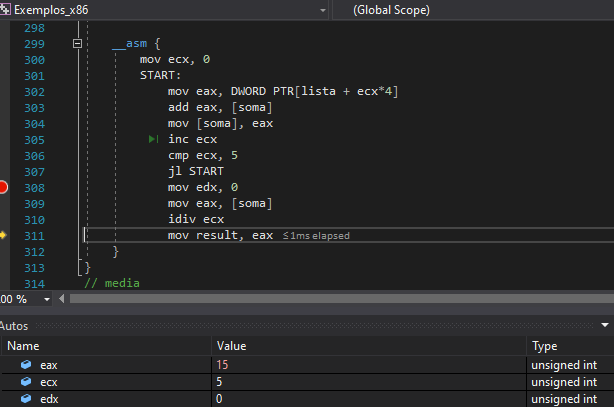
idiv ecx

mov result, eax

}

}

// media



Soma todos os elementos do vetor e divide pela quantidade do mesmo.

void Exercicio5(void) {

int lista[] = { 5, 30, 15, 20, 25 };

int maior;

int cont = 0;

\_\_asm {

mov ecx, 0

mov eax, DWORD PTR[lista + ecx\*4]

mov[maior], eax

inc ecx

INICIO :

mov eax, [maior]

cmp eax, DWORD PTR[lista + ecx\*4]

jl MAIORTRUE

jg MAIORFALSE

MAIORTRUE :

mov eax, DWORD PTR[lista + ecx\*4]

mov[maior], eax

MAIORFALSE :

inc ecx

inc cont

cmp cont, 5

jl INICIO

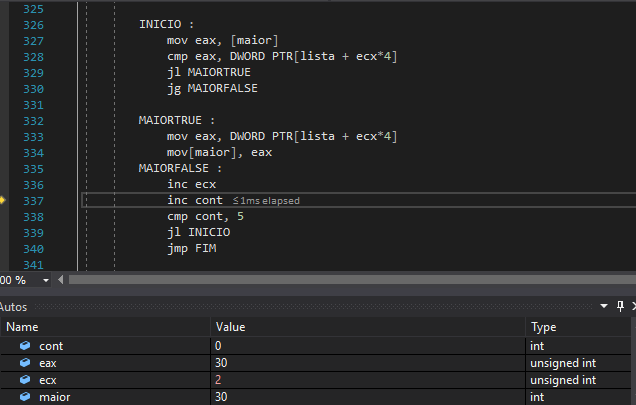
jmp FIM

FIM :

}

}

// maior valor



Percorrer a lista até encontrar um valor maior do que o anterior encontrado. E quando encontrar, substituir o valor.

void Exercicio6(void) {

//só funcionará para valores maiores ou iguais a 2

int valor = 83;

int incremento = 2;

int isPrime = 0;

\_\_asm {

mov edx, 0

mov eax, [valor]

idiv [incremento]

inc [incremento]

cmp edx, 0

je FIMFALSE

START :

mov eax, valor

cmp [incremento], eax

jge FIMTRUE

mov edx, 0

mov eax, [valor]

idiv[incremento]

cmp edx, 0

je FIMFALSE

add [incremento], 2

jmp START

FIMTRUE:

mov [isPrime], 1

jmp FIM

FIMFALSE:

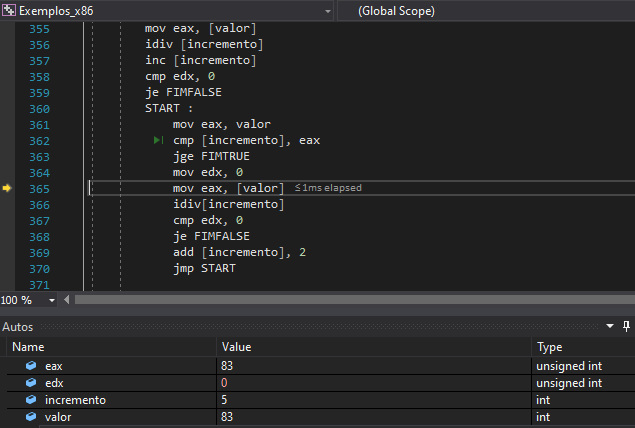
mov [isPrime], 0

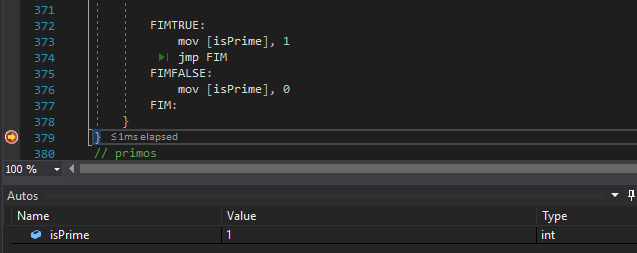
FIM:

}

}

// primos



Compara se o valor é divisível por outro número além dele e pelo número 1 (definição de primo) 

Caso o valor da variável “isPrime” for 1, ele é primo. Se não for primo, o valor será igual a 0.