

```

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

```

The screenshot displays a debugger interface with three main sections:

- Assembly View (Top):** Shows assembly instructions corresponding to the C code. The current instruction is at line 251: `cmp ecx, 6`. The instruction at line 256, `inc cont`, is highlighted with a yellow background, indicating a breakpoint or the current execution point.
- Autos Pane (Middle):** A table showing the current state of local variables:

Name	Value	Type
eax	2	unsigned int
ecx	2	unsigned int
valor	1	int
- Assembly View (Bottom):** Another instance of the assembly code, showing the instruction at line 256: `inc cont` with a value of 1ms elapsed.

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

```

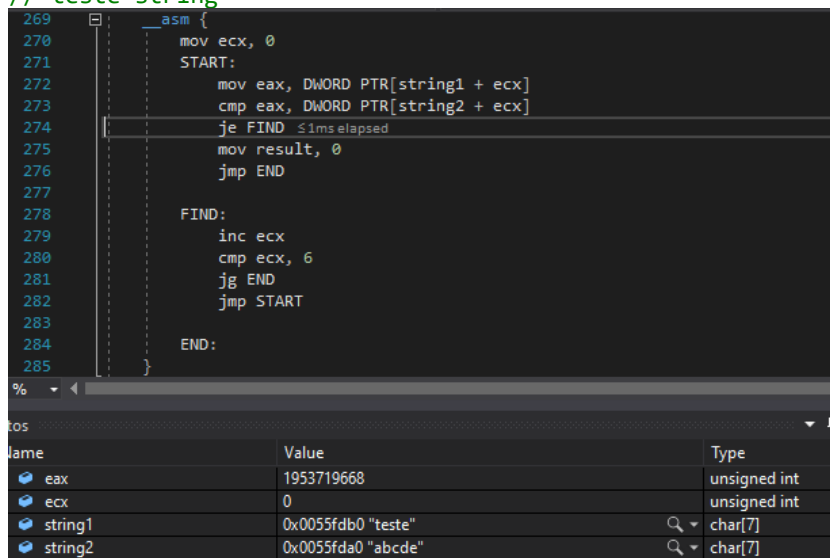
void Exercicio2(void) {
    char string1[] = "teste\0";
    char string2[] = "abcde\0";
    bool result = true;

    __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

```



The screenshot shows a debugger window with assembly code on the left and a register window on the right. The assembly code is as follows:

```

269  __asm {
270      mov ecx, 0
271      START:
272          mov eax, DWORD PTR[string1 + ecx]
273          cmp eax, DWORD PTR[string2 + ecx]
274          je FIND
275          mov result, 0
276          jmp END
277
278      FIND:
279          inc ecx
280          cmp ecx, 6
281          jg END
282          jmp START
283
284      END:
285  }

```

The register window shows the following values:

Name	Value	Type
eax	1953719668	unsigned int
ecx	0	unsigned int
string1	0x0055fdb0 "teste"	char[7]
string2	0x0055fda0 "abcde"	char[7]

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
```

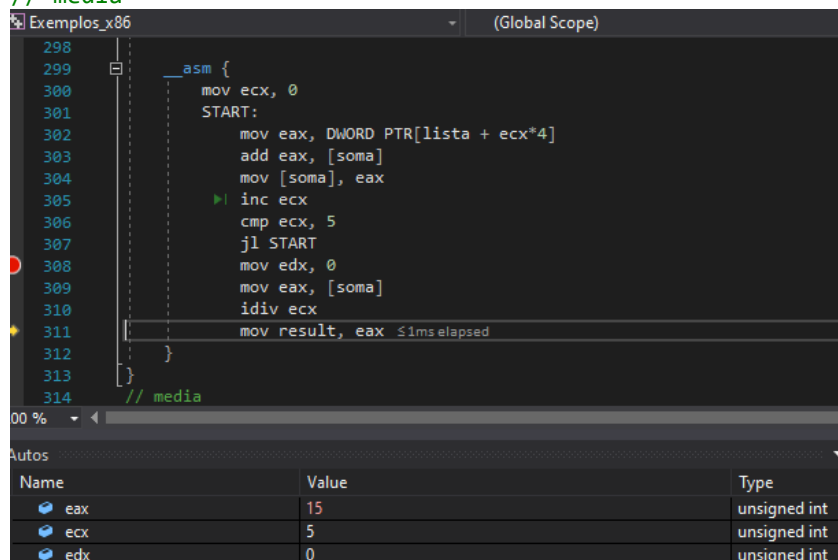
Exercício de número 3 não completo pela falta de conhecimento na hora de retirar o valor da lista.

```

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

```



```

298
299
300     __asm {
301         mov ecx, 0
302         START:
303             mov eax, DWORD PTR[lista + ecx*4]
304             add eax, [soma]
305             mov [soma], eax
306             inc ecx
307             cmp ecx, 5
308             jl START
309             mov edx, 0
310             mov eax, [soma]
311             idiv ecx
312             mov result, eax
313     }
314 // media

```

Name	Value	Type
eax	15	unsigned int
ecx	5	unsigned int
edx	0	unsigned int

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

325	
326	INICIO :
327	mov eax, [maior]
328	cmp eax, DWORD PTR[lista + ecx*4]
329	jl MAIORTRUE
330	jg MAIORFALSE
331	
332	MAIORTRUE :
333	mov eax, DWORD PTR[lista + ecx*4]
334	mov[maior], eax
335	MAIORFALSE :
336	inc ecx
337	inc cont ≤ 1ms elapsed
338	cmp cont, 5
339	jl INICIO
340	jmp FIM
341	

Name	Value	Type
cont	0	int
eax	30	unsigned int
ecx	2	unsigned int
maior	30	int

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

```

The screenshot shows a debugger window titled 'Exemplos_x86' with '(Global Scope)' selected. The assembly window displays the following instructions:

```

355 mov eax, [valor]
356 idiv [incremento]
357 inc [incremento]
358 cmp edx, 0
359 je FIMFALSE
360 START :
361 mov eax, valor
362 cmp [incremento], eax
363 jge FIMTRUE
364 mov edx, 0
365 mov eax, [valor]
366 idiv[incremento]
367 cmp edx, 0
368 je FIMFALSE
369 add [incremento], 2
370 jmp START
371

```

The CPU registers window shows the following values:

Name	Value	Type
eax	83	unsigned int
edx	0	unsigned int
incremento	5	int
valor	83	int

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

```
371
372      FIMTRUE:
373          mov [isPrime], 1
374          ►| jmp FIM
375      FIMFALSE:
376          mov [isPrime], 0
377      FIM:
378  }
379  } 51ms elapsed
380  // primos
```

100 %

Autos

Name	Value	Type
isPrime	1	int

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