Comparando C/C++ e Assembly

Aluno: Roberto Alves Neto

Comparando C e C++:

Em programas simples a diferença entre as duas linguagens é pouca.

Entre elas está que em C++ temos uma maior facilidade de receber e exibir valores e podemos usar orientação a objetos.

Comparando C e C++:

Código em C

```c
#include <stdio.h>

int main()
{
    int val1, val2;
    scanf("%d", &val1);
    scanf("%d", &val2);
    printf("%d", (val1 + val2)/2);

    return 0;
}
```

Código em C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    int val1, val2;
    cin >> val1;
    cin >> val2;
    cout << ((val1 + val2)/2);
}
```

Assembly:

Por ser próximo ao nível de máquina a complexidade aumenta bastante, dentre os principais fatores temos:

- As instruções mudam conforme o compilador, a arquitetura e o processador;
- Lidamos diretamente com os registradores, não podendo realizar as operações entre as variáveis;
- Tudo é na tabela ASCII, precisando realizar conversões;
- A realização de cálculos só funcionam para um dígito (8bits), para realizar cálculos com valores maiores precisamos criar loops e salvar a sobra em pilhas;
- Loops e condições são feitos com saltos, semelhantes ao GoTo de C, o que dificulta o entendimento;
- O código cresce rapidamente e é repetitivo.

Exemplificando dificuldades citadas:

As instruções mudam conforme o compilador, a arquitetura e o processador:

Usei o SYS_READ e STDIN para guardar os valores de chamada ao sistema para simplificar. Em 32bits o SYS_EXIT é 1, já em 64bits é 60 e ao invés de usarmos int 0x80 para chamar a parada do sistema, usamos syscall.

```
;entrada 1
mov eax, SYS_READ
mov ebx, STDIN
mov ecx, num1
mov edx, 2
int 0×80
```

```
; get first number
LEA DX, MSGA
MOV AH, 09h
INT 21h
```

Exemplificando dificuldades citadas:

Lidamos diretamente com os registradores, não podendo realizar as operações entre as variáveis:

```
mov     al, [num1]
adc     al, [num2]
```

Tudo é na tabela ASCII, precisando realizar conversões:

```
;destransforma de ascii
mov eax, [num1]
sub eax, '0'
mov ebx, [num2]
sub ebx, '0'
```

Exemplificando dificuldades citadas:

A realização de cálculos só funcionam para um dígito (8bits), para realizar cálculos com valores maiores precisamos criar loops e salvar a sobra em pilhas:

```
add_loop:
    mov     al, [num1 + esi]
    adc     al, [num2 + esi]
    aaa
    pushf
    or  al, 30h
    popf

    mov [sum + esi], al
    dec esi
    loop    add_loop
```

Exemplificando dificuldades citadas:

Loops e condições são feitos com saltos, semelhantes ao GoTo de C, o que dificulta o entendimento:

```
L1:
    ;incrementa
    inc al
    mov [res], al

    ;se al ⩾ ah, para loop
    cmp al, ah
    jge Gotoo

    ;devolve os valores aos registradores
    mov al, [res]
    mov ah, [num2]

    ;pulo
    jmp L1
Gotoo:
```

Exemplificando dificuldades citadas:

O código cresce rapidamente e é repetitivo:

Receber uma entrada; printar uma saída; tratamentos para números, são todas instruções grandes e que aparecem com frequência que poderiam ser simplificadas.

```
;entrada 1
mov eax, SYS_READ
mov ebx, STDIN
mov ecx, num1
mov edx, 2
int 0×80
```

```
;entrada 2
mov eax, SYS_READ
mov ebx, STDIN
mov ecx, num2
mov edx, 2
int 0×80
```

```
;print resp
mov eax, SYS_WRITE
mov ebx, STDOUT
mov ecx, res
mov edx, 1
int 0×80
```