Entrada e Saída

João Marcelo Uchôa de Alencar

21 de março de 2023

1/23

2/23

Introdução

- E/S (entrada e saída) na linguagem *assembly* pode ser bem complicada.
- É possível acessar as funções de E/S em C utilizá-las enquanto aprendemos outros conceitos em *assembly*.
- Vamos começar mostrando E/S com inteiros, mas pontos flutuantes são basicamente a mesma coisa.
- A E/S em 64 *bits* já é diferente, mas veremos adiante.

3/23

Olá Mundo em C

- Programa teste clássico.
- É importante pois valida o funcionamento correto do compilador.
- Versão em C:

```c
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    return 0;
}
```

- No Windows com o Visual Studio Community instalado, para compilar no *PowerShell*:

```
PS C:\...\HelloWord> cl.exe \HelloWorld.c
```

- Os arquivos *HelloWorld.obj* e *HelloWorld.exe* são gerados na mesma pasta.

4/23

Olá Mundo em *Assembly*

```
        .686
        .model flat, c
        .stack 100h
printf  PROTO   arg1:Ptr Byte
        .data
msg1    byte    "Hello World!",0Ah,0
        .code
main    proc
        INVOKE  printf, ADDR msg1
        ret
main    endp
        end
```

- Compilar no *PowerShell* (em uma linha só):

```
PS C:\...\HelloWord> ml /Cx /coff HelloWorld.asm
/link /SUBSYSTEM:console /out:HelloWorld.exe
kernel32.lib legacy_stdio_definitions.lib msvcrt.lib
```


5/23

Olá Mundo em *Assembly*

- A diretiva PROTO indica o protótipo da função *printf*. O ligador (*linker*) irá procurá-la nas bibliotecas do sistema para formar o executável.
- O parâmetro *arg1:Ptr Byte* indica que *printf* espera receber um cadeia de *bytes*.
- A diretiva INVOKE invoca a função, como um subprograma, limpando os registradores *eax*, *ecx* e *edx* para utilizá-los na passagem de parâmetros.
- *ADDR msg1* indica o endereço da *string* a ser impressa:
  - A mensagem está na seção *.data*.
  - *0Ah* é o hexadecimal para $\backslash n$.
  - *Strings* são terminadas com 0.

6/23

printf com *string* de formatação

```c
#include <stdio.h>
int main() {
   printf("%s\n", "Hello World!");
   return 0;
}
```

- A formatação é separada dos dados.
- Usando o símbolo %, podemos construir a *string* e depois definir as variáveis e valores.

7/23

printf com *string* de formatação

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
                .data
msg1fmt         byte "%s", 0Ah, 0
msg1            byte "Hello World!", 0
                .code
main            proc
                INVOKE printf, ADDR msg1fmt, ADDR msg1
                ret
main            endp
                end
```


8/23

printf com *string* de formatação

- A sentença PROTO tem um argumento adicional, *printlist: VARARG*.
- A formatação, %s, é declarada em uma variável diferente (*msg1fmt*) da variável dos dados (*msg1*).
- Ambas são *strings*, precisam ser terminadas em 0.
- A diretiva INVOKE referencia dois endereços agora.

9/23

Saída de Inteiros

```c
#include <stdio.h>

int main() {
    int number;
    number = 5;
    printf("%s%d\n", "The number is: ", number);
    return 0;
}
```


10/23

Saída de Inteiros

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
                .data
msg1fmt         byte "%s%d", 0Ah, 0
msg1            byte "The number is:", 0
number          sdword ?
                .code
main            proc
                mov number, 5
                INVOKE printf, ADDR msg1fmt, ADDR msg1, number
                ret
main            endp
                end
```


11/23

Saída de Inteiros

- *msg1fmt* agora tem %d adicionado.
- *number* foi declarada como *sdword*.
- *number* também foi incluída na diretiva INVOKE.
- Veja que para acessar o primeiro endereço das *strings*, precisamos da diretiva ADDR.
- No caso de *number*, basta o nome da variável.

12/23

Saída de Inteiros

```c
#include <stdio.h>

int main() {
    int num1 = 5, num2 = 7;
    number = 5;
    printf("\n%d%s%d\n\n", num1, " is not equal to ", num2);
    return 0;
}
```


13/23

Saída de Inteiros

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
                .data
msg1fmt         byte 0Ah,"%d%s%d", 0Ah, 0Ah, 0
msg1            byte " is not equal to ", 0
num1            sdword 5
num2            sdword 7
                .code
main            proc
                INVOKE printf, ADDR msg1fmt, num1, ADDR msg1, num2
                ret
main            endp
                end
```


14/23

Saída de Inteiros

- Nenhuma mudança na diretiva PROTO.
- INVOKE agora utiliza quatro argumentos.
- VARARG permite número de argumentos variável.
- Inicializamos *num1* e *num2* para evitar as instruções *mov*.

15/23

Entrada de Inteiros

```c
#include <stdio.h>

int main() {
    int number;
    scanf("%d", &number);
    printf("\n%s%d\n\n", "The number is:", number);
    return 0;
}
```

O & indica o endereço de *number* que deve ser passado a *scanf* para que o valor lido do teclado seja colocado na memória.

16/23

Entrada de Inteiros

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
scanf           PROTO arg2:Ptr Byte, inputlist:VARARG
                .data
in1fmt          byte "%d", 0
msg1fmt         byte 0Ah,"%s%d", 0Ah, 0Ah, 0
msg1            byte "The number is:", 0
number          sdword ?
                .code
main            proc
                INVOKE scanf, ADDR in1fmt,  ADDR number
                INVOKE printf, ADDR msg1fmt, ADDR msg1, number
                ret
main            endp
                end
```


17/23

Entrada de Inteiros

- Ainda precisamos da *string* de formatação em *scanf*, %d indica que um inteiro será lido.
- Para desempenhar o papel do & em C, usamos a diretiva ADDR.
- Novamente, como *printf* não precisa do ponteiro, usamos apenas *number*.
- Esse programa é bem limitado, o usuário digita um número e o mesmo é repetido na tela.

18/23

Entrada de Inteiros

```c
#include <stdio.h>

int main() {
    int number;
    printf("\n%s", "Enter an integer: ");
    scanf("%d", &number);
    printf("\n%s%d\n\n", "The number is:", number);
    return 0;
}
```


19/23

Entrada de Inteiros

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
scanf           PROTO arg2:Ptr Byte, inputlist:VARARG
                .data
in1fmt          byte "%d", 0
msg0fmt         byte 0Ah,"%s", 0
msg1fmt         byte 0Ah,"%s%d", 0Ah, 0Ah, 0
msg0            byte "Enter an integer: ", 0
msg1            byte "The number is: " , 0
number          sdword ?
                .code
main            proc
                INVOKE printf, ADDR msg0fmt, ADDR msg0
                INVOKE scanf, ADDR in1fmt,  ADDR number
                INVOKE printf, ADDR msg1fmt, ADDR msg1, number
                ret
main            endp
                end
```


20/23

Entrada de Inteiros

```c
#include <stdio.h>

int main() {
    int num1, num2;
    printf("\n%s", "Enter an integer for num1: ");
    scanf("%d", &num1);
    num2 = num1;
    printf("\n%s%d\n\n","The integer in num2 is: ", num2);
}
```


21/23

Entrada de Inteiros

```
                .686
                .model flat, c
                .stack 100h
printf          PROTO arg1:Ptr Byte, printlist:VARARG
scanf           PROTO arg2:Ptr Byte, inputlist:VARARG
                .data
in1fmt          byte "%d", 0
msg0fmt         byte 0Ah,"%s", 0
msg1fmt         byte 0Ah,"%s%d", 0Ah, 0Ah, 0
msg0            byte "Enter an integer for num1: ", 0
msg1            byte "The integer in num2 is: " , 0
num1            sdword ?
num2            sdword ?
                .code
main            proc
                INVOKE printf, ADDR msg0fmt, ADDR msg0
                INVOKE scanf, ADDR in1fmt,  ADDR num1
                mov eax, num1
                mov num2, eax
                INVOKE printf, ADDR msg1fmt, ADDR msg1, num2
                ret
main            endp
                end
```


22/23

Resumo

- Use as diretivas PROTO e INVOKE para invocar as funções da biblioteca C *printf* e *scanf*.
- A diretiva INVOKE destrói o conteúdo dos registradores *eax*, *ecx* e *edx*.
- Usar *strings* de formatação ajudam a tornar o código mais legível.
- O *0Ah* em *assembly* equivale ao n em C, ao *endl* do C++ e *println* do Java.
- Sempre finalize *strings* com 0.
- Na saída, não esqueça a diretiva ADDR para *strings*.
- Na entrada, não esqueça do ADDR para todas as variáveis.

23/23