

Programación Assembly x86 - Convención C

Organización del Computador II

29 de agosto de 2023

En este taller vamos a trabajar con código C y ASM para ejercitar el uso y adhesión a contratos estructurales y comportamentales. El checkpoint 1 se trata de un repaso de los conceptos vistos en clase, necesarios para encarar la programación en forma efectiva. El resto de los checkpoints se resuelven programando determinadas rutinas en assembly.

Al corregir cada **checkpoint** todos los miembros del grupo deben estar presentes, salvo aquellas instancias en que puedan justificar su ausencia previamente. Si al finalizar la práctica de la materia algún miembro cuenta con mas de un 20 % de ausencia no justificada en la totalidad de los checkpoints se considerará la cursada como desaprobada.

1. Conceptos generales

En este checkpoint deberán responder algunas preguntas conceptuales relacionadas a lo que vimos en las clases teóricas y en la presentación de hoy. Las preguntas tienen que ver con **alineación de datos, convención de llamada y bibliotecas compartidas**.

- ¿Qué entienden por convención de llamada? ¿Cómo está definida en la ABI de System V para 64 y 32 bits?
- ¿Quién toma la responsabilidad de asegurar que se cumple la convención de llamada en C? ¿Quién toma la responsabilidad de asegurar que se cumple la convención de llamada en ASM?
- ¿Qué es un stack frame? ¿A qué se le suele decir **prólogo y epílogo**?
- ¿Cuál es el mecanismo utilizado para almacenar **variables temporales**?
- ¿A cuántos bytes es necesario alinear la pila si utilizamos funciones de `libc`? ¿Si la pila está alineada a 16 bytes al realizarse una llamada función, cuál va a ser su alineamiento al ejecutar la primera instrucción de la función llamada?
- Una actualización de bibliotecas realiza los siguientes cambios a distintas funciones. ¿Cómo se ven impactados los programas ya compilados?

Sugerencia: Describan la convención de llamada de cada una (en su versión antes y después del cambio).

- Una biblioteca de procesamiento cambia la estructura `pixel_t`:
 - Antes era `struct { uint8_t r, g, b, a; }`
 - Ahora es `struct { uint8_t a, r, g, b; }`¿Cómo afecta esto a la función `void a_escalado_grises(uint32_t ancho, uint32_t alto, pixel_t* data)`?
- Se reordenan los parámetros (i.e. intercambian su posición) de la función `float sumar_floats(float* array, uint64_t tamano)`.
- La función `uint16_t registrar_usuario(char* nombre, char* contraseña)` registra un usuario y devuelve su ID. Para soportar más usuarios se cambia el tipo de retorno por `uint64_t`.
- La función `void cambiar_nombre(uint16_t user_id, char* nuevo_nombre)` también recibe la misma actualización. ¿Qué sucede ahora?
- Se reordenan los parámetros de `int32_t multiplicar(float a, int32_t b)`.

Una vez analizados los casos específicos describan la situación general: ¿Qué sucede si una función externa utilizada por nuestro programa¹ cambia su interfaz (parámetros o tipo devuelto) luego de una actualización?

Checkpoint 1

¹Es decir, que vive en una **biblioteca compartida**

2. C/ASM pasaje de parámetros

En este checkpoint y en los que siguen vamos a desarrollar funciones en **ASM** que son llamadas desde código **C** haciendo uso de la convención de llamada de 64 bits. Pueden encontrar el código vinculado a este checkpoint en **checkpoint2.asm** y **checkpoint2.c**, la declaración de las funciones se encuentra en **checkpoints.h**. También ahí mismo se explica qué debe hacer dicha función.

Programen en assembly las siguientes funciones:

- a) `alternate_sum_4`,
- b) `alternate_sum_4_using_c`
- c) `alternate_sum_4_simplified`
- d) `alternate_sum_8`
- e) `product_2_f`
- f) `product_9_f`

El archivo que tienen que editar es **checkpoint2.asm**. Para todas las declaraciones de las funciones en asm van a encontrar la firma de la función. Completar para cada parámetro en qué registro o posición de la pila se encuentra cada uno.

La siguiente información aplica para este checkpoint y los siguientes

Compilación y Testeo

El archivo **main.c** es para que ustedes realicen pruebas básicas de sus funciones. Sientanse a gusto de manejarlo como crean conveniente. Para compilar el código y poder correr las pruebas cortas implementadas en main deberá ejecutar **make main** y luego **./runMain.sh**.

En cambio, para compilar el código y correr las pruebas intensivas deberá ejecutar **./runTester.sh**. El programa puede correrse con **./runMain.sh** para verificar que no se pierde memoria ni se realizan accesos incorrectos a la misma.

Pruebas intensivas (Testing)

Entregamos también una serie de *tests* o pruebas intensivas para que pueda verificarse el buen funcionamiento del código de manera automática. Para correr el testing se debe ejecutar **./runTester.sh**, que compilará el *tester* y correrá todos los tests de la cátedra. Luego de cada test, el *script* comparará los archivos generados por su TP con las soluciones correctas provistas por la cátedra. También será probada la correcta administración de la memoria dinámica.

Checkpoint 2

3. Recorrido de estructuras C/ASM

En este checkpoint vamos implementar código vinculado al acceso de estructuras. En particular, vamos a usar dos estructuras muy similares a `lista_t` del taller anterior y vamos a implementar en `asm` la función que contaba la cantidad total de elementos de la lista, para ambas estructuras. Pueden encontrar el código vinculado a este checkpoint en `checkpoint3.c` y `checkpoint3.asm`.

Las definiciones de las estructuras las pueden encontrar en el archivo `checkpoints.h`. Las de los nodos correspondientes son las siguientes:

```
typedef struct nodo_s {
    struct nodo_s* next;    // Siguiente elemento de la lista o NULL si es el final
    uint8_t categoria;      // Categoría del nodo
    uint32_t* arreglo;      // Arreglo de enteros
    uint32_t longitud;      // Longitud del arreglo
} nodo_t;

typedef struct __attribute__((__packed__)) packed_nodo_s {
    struct packed_nodo_s* next;    // Siguiente elemento de la lista o NULL si es el final
    uint8_t categoria;            // Categoría del nodo
    uint32_t* arreglo;            // Arreglo de enteros
    uint32_t longitud;            // Longitud del arreglo
} packed_nodo_t;
```

Programen en assembly las funciones:

- a) `uint32_t` cantidad_total_de_elementos(`lista_t*` lista),
- b) `uint32_t` cantidad_total_de_elementos_packed(`packed_lista_t*` lista)

El archivo que tienen que editar es `checkpoint3.asm`. Para todas las declaraciones de las funciones en `asm` van a encontrar la firma de la función. Completar para cada parámetro en qué registro o posición de la pila se encuentra cada uno.

Checkpoint 3

4. Memoria dinámica con strings

En este checkpoint vamos implementar código vinculado para el manejo de strings. Recuerden que C representa a los strings como una sucesión de `char`'s terminada con el caracter nulo `'\0'`. Pueden encontrar el código vinculado a este checkpoint en `checkpoint4.c` y `checkpoint4.asm`. Recuerden que para realizar el clonado de un string, van a tener que usar memoria dinámica. Luego analizaremos el uso de la pila.

a) Programen en assembly las siguientes funciones. El archivo que tienen que editar es `checkpoint4.asm`.

- a) `strlen`
- b) `strPrint`
- c) `strClone`
- d) `strCmp`
- e) `strdelete`

b) Se proveen resueltas las funciones `stack_snooper` y `stack_snooper_n` en el archivo `snooper.asm`. Se pide:

- a) Analizar el código de `stack_snooper` y `stack_snooper_n` y explicar qué y cómo lo hacen.
- b) En las funciones `strlen`, y `strClone` pushear al stack (en el prólogo) algún valor conocido que sirva como delimitador. Luego, en `strlen` invocar a `stack_snooper_n` con un `n=1` y `n=2`. Compilar, ejecutar y explicar el contenido de la salida.

Checkpoint 4