Práctica de Organización del Computador II

ABI

Primer Cuatrimestre 2024

Organización del Computador II
DC - UBA

Contratos de función

Contratos de funciones

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos.**

Contratos de funciones

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos.**
Es decir, tenemos ciertas garantías sobre cómo se ubican los datos en memoria (*endianness*, *layouts* de structs, arrays, etc)

Contratos de funciones

En la clase pasada presentamos la alineación de los datos en memoria como un tipo de **contrato de datos.**
Es decir, tenemos ciertas garantías sobre cómo se ubican los datos en memoria (*endianness*, *layouts* de structs, arrays, etc)

Veamos ahora la forma en la que definimos y nos adherimos a un **contrato de función**.

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.
- Para llamar a `product` hay que proporcionar dos valores: uno de tipo `int32_t*` y otro de tipo `uin32_t`.

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.
- Para llamar a `product` hay que proporcionar dos valores: uno de tipo `int32_t*` y otro de tipo `uin32_t`.

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.
- Para llamar a `product` hay que proporcionar dos valores: uno de tipo `int32_t*` y otro de tipo `uin32_t`.

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Existe una función llamada `product`.
- Quien llame a `product` va a obtener un valor de tipo `int32_t`.
- Para llamar a `product` hay que proporcionar dos valores: uno de tipo `int32_t*` y otro de tipo `uin32_t`.

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).
- Todo uso o implementación de la función **deben coincidir en tipo devuelto, cantidad y tipo de parámetros**.
- ¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).
- Todo uso o implementación de la función **deben coincidir en tipo devuelto, cantidad y tipo de parámetros**.
- ¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

Contratos de funciones

¿Qué es una declaración de función en C?

```
int32_t product(int32_t *arr, uint32_t length);
```

- Este contrato es respetado automáticamente por el compilador de C (gcc, clang, etc).
- Todo uso o implementación de la función **deben coincidir en tipo devuelto, cantidad y tipo de parámetros**.
- ¿Qué sucede cuando queremos llamar a una función de C desde ASM o a una función de ASM desde C?

Contratos de funciones

¿Qué sucede cuando queremos llamar a una función de `C` desde `ASM` o a una función de `ASM` desde `C`?

Contratos de funciones

¿Qué sucede cuando queremos llamar a una función de `C` desde `ASM` o a una función de `ASM` desde `C`?

Respuesta: Vamos a tener que definir el alcance de nuestro contrato en términos de la plataforma particular.

Contratos de funciones

¿Qué sucede cuando queremos llamar a una función de `C` desde `ASM` o a una función de `ASM` desde `C`?

Respuesta: Vamos a tener que definir el alcance de nuestro contrato en términos de la plataforma particular.

Corolario: Los contratos de función en un lenguaje de alto nivel se pueden definir independientemente de la plataforma.

Contratos de funciones

¿Entonces cualquier lenguaje que genere **código objeto y respete el contrato de función** puede interactuar con funciones ubicadas en bibliotecas binarias (código objeto) que adhieran al contrato?

Contratos de funciones

¿Entonces cualquier lenguaje que genere **código objeto y respete el contrato de función** puede interactuar con funciones ubicadas en bibliotecas binarias (código objeto) que adhieran al contrato?

Respuesta: Si, para eso vamos a tener que hablar de la **ABI (Application Binary Interface)**

Interfaz binaria de aplicación (ABI)

Interfaz binaria de aplicación

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.

Interfaz binaria de aplicación

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.
Estos contratos específicos se llamarán **Interfaces Binarias de Aplicación (ABIs)** y definen la forma en que las funciones serán llamadas, cómo se pasan los parámetros y que invariantes estructurales deben hacerse valer.

Interfaz binaria de aplicación

Cuando queremos exponer una interfaz parecida a quien desarrolla código en **bajo nivel** vamos a tener que definir **contratos específicos para la plataforma**.
Estos contratos específicos se llamarán **Interfaces Binarias de Aplicación (ABIs)** y definen la forma en que las funciones serán llamadas, cómo se pasan los parámetros y que invariantes estructurales deben hacerse valer.
La ABI no sólo depende del procesador, el sistema operativo también juega un rol.

Interfaz binaria de aplicación

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Interfaz binaria de aplicación

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Interfaz binaria de aplicación

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Interfaz binaria de aplicación

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Interfaz binaria de aplicación

Una **ABI** va a definir contratos sobre otros elementos que no nos interesan en este momento, como por ejemplo:

- Formato de archivos objeto y ejecutables
- Uso de bibliotecas compartidas
- Parámetros pasados al proceso
- Ubicación de tablas globales del sistema

Interfaz binaria de aplicación

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de usuarix (**Convención de llamada**)

Interfaz binaria de aplicación

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de usuarix (**Convención de llamada**)

Interfaz binaria de aplicación

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de usuarix (**Convención de llamada**)

Interfaz binaria de aplicación

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de usuarix (**Convención de llamada**)

Interfaz binaria de aplicación

Si bien son importantes no vamos a hablar de esos contratos en esta materia. Los que sí nos importan son:

- El set de instrucciones
- Tamaño y alineación de los tipos de datos primitivos
- La forma de enviar y recibir información usando funciones del sistema (Convención de llamada para **System Calls**)
- La forma de enviar y recibir información usando funciones de usuarix (**Convención de llamada**)

Convención de llamadas

Pregunta: ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

Convención de llamadas

Pregunta: ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

Convención de llamadas

Pregunta: ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

Convención de llamadas

Pregunta: ¿Cómo compartimos información entre funciones consistentemente y a nivel binario?

- ¿Usamos registros?
- ¿Usamos la pila?

Respuesta: Sí (a todo)

Interfaz binaria de aplicación

Pregunta: ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

- **Volátiles**: La función llamada no tiene obligación de conservarlos
- **No volátiles**: Si la función llamada los cambia debe restaurarlos antes del `return`

Interfaz binaria de aplicación

Pregunta: ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

Respuesta: Vamos a dividir los registros entre volátiles y no volátiles

- **Volátiles**: La función llamada no tiene obligación de conservarlos
- **No volátiles**: Si la función llamada los cambia debe restaurarlos antes del `return`

Interfaz binaria de aplicación

Pregunta: ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

Respuesta: Vamos a dividir los registros entre volátiles y no volátiles

- **Volátiles**: La función llamada no tiene obligación de conservarlos
- **No volátiles**: Si la función llamada los cambia debe restaurarlos antes del `return`

Pregunta: ¿Qué pasa con los registros en uso al realizar una llamada? ¿Conservan sus valores al regresar?

Respuesta: Vamos a dividir los registros entre volátiles y no volátiles

- **Volátiles**: La función llamada no tiene obligación de conservarlos
- **No volátiles**: Si la función llamada los cambia debe restaurarlos antes del `return`

ABI: System V

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila**.
- Otro para 32 bits, **que sólo utiliza la pila**.

- En `x86-64/Linux` (64bits) se denomina `System V AMD64 ABI`
- En `x86/Linux` (32bits) se conoce como `System V i386 ABI`

ABI: System V

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila**.
- Otro para 32 bits, **que sólo utiliza la pila**.

- En `x86-64/Linux` (64bits) se denomina `System V AMD64 ABI`
- En `x86/Linux` (32bits) se conoce como `System V i386 ABI`

ABI: System V

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila**.
- Otro para 32 bits, **que sólo utiliza la pila**.

- En `x86-64/Linux` (64bits) se denomina `System V AMD64 ABI`
- En `x86/Linux` (32bits) se conoce como `System V i386 ABI`

ABI: System V

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila**.
- Otro para 32 bits, **que sólo utiliza la pila**.

Las convenciones dependen de la arquitectura del procesador y del sistema operativo:

- En `x86-64/Linux` (64bits) se denomina `System V AMD64 ABI`
- En `x86/Linux` (32bits) se conoce como `System V i386 ABI`

ABI: System V

La **ABI** que utilizaremos define dos convenciones de llamada:

- Uno para 64 bits, **que utiliza los registros de propósito general y la pila**.
- Otro para 32 bits, **que sólo utiliza la pila**.

Las convenciones dependen de la arquitectura del procesador y del sistema operativo:

- En `x86-64/Linux` (64bits) se denomina `System V AMD64 ABI`
- En `x86/Linux` (32bits) se conoce como `System V i386 ABI`

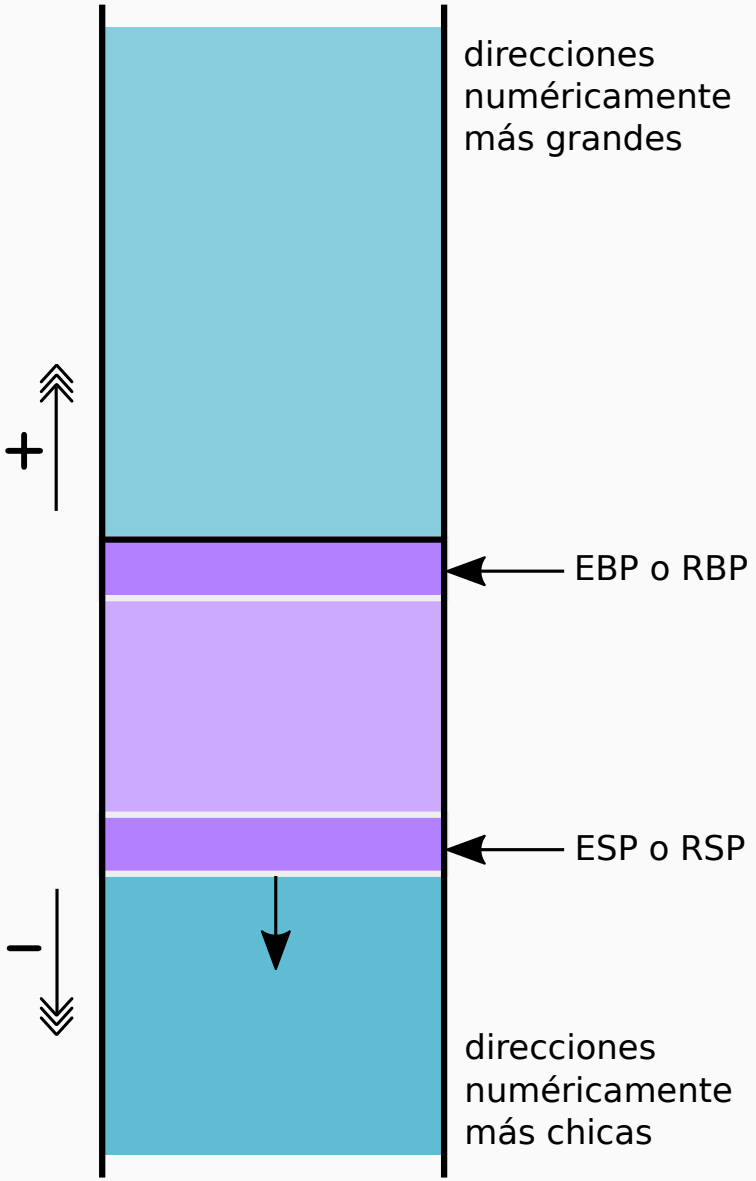
Vamos a usar el primero ahora haciendo programación de aplicaciones y el segundo cuando hagamos programación de sistemas.

Uso de la pila

La pila

- La pila es una estructura en memoria
- Sirve para guardar información **local** a una función
- Contiene información de **contexto**: parámetros, dirección de retorno

Uso de la pila



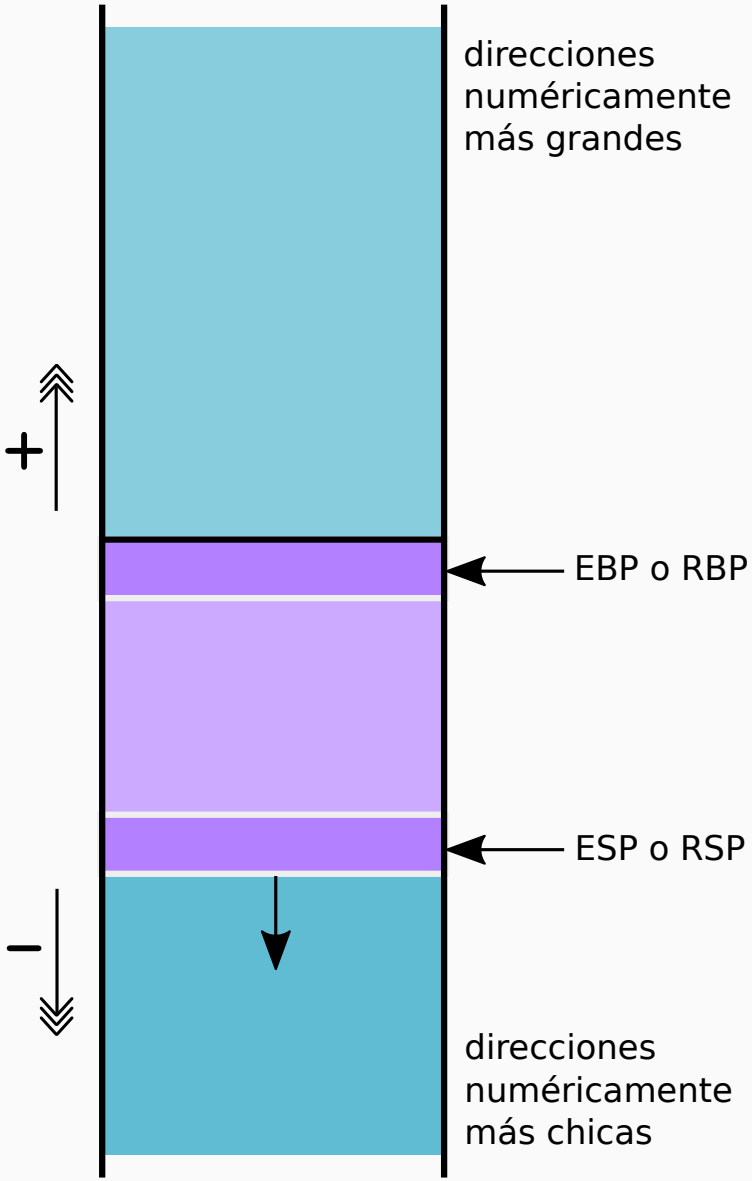
En 32 bits

- Los registros EBP y ESP
- EBP (Base Pointer) apunta a la base
- ESP (Stack Pointer) al tope (último elemento válido)

En 64 bits

- Los registros RBP y RSP
- RBP (Base Pointer) apunta a la base
- RSP (Stack Pointer) al tope (último elemento válido)

Uso de la pila



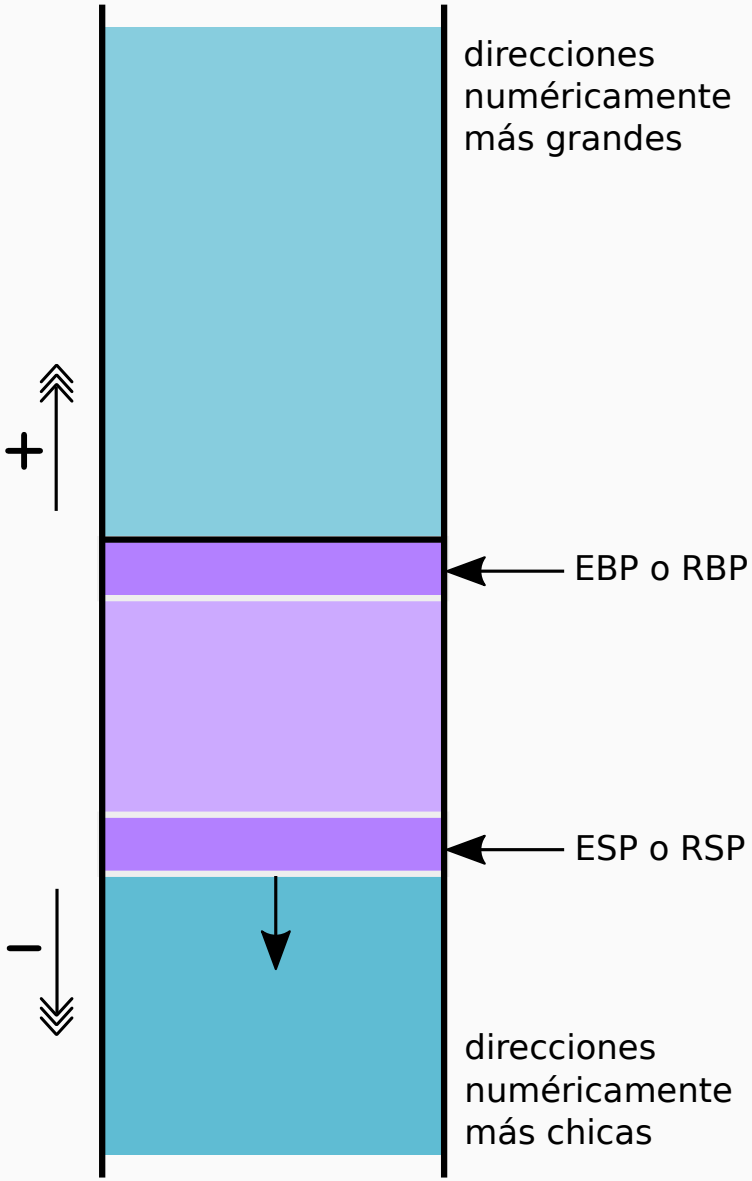
En 32 bits

- Los registros EBP y ESP
- EBP (Base Pointer) apunta a la base
- ESP (Stack Pointer) al tope (último elemento válido)

En 64 bits

- Los registros RBP y RSP
- RBP (Base Pointer) apunta a la base
- RSP (Stack Pointer) al tope (último elemento válido)

Uso de la pila



En 32 bits

- Los registros `EBP` y `ESP`
- `EBP` (Base Pointer) apunta a la base
- `ESP` (Stack Pointer) al tope (último elemento válido)

En 64 bits

- Los registros `RBP` y `RSP`
- `RBP` (Base Pointer) apunta a la base
- `RSP` (Stack Pointer) al tope (último elemento válido)

Uso de la pila

Alineación

- 4 bytes (en 32bits)
- 16 bytes para llamar funciones de C

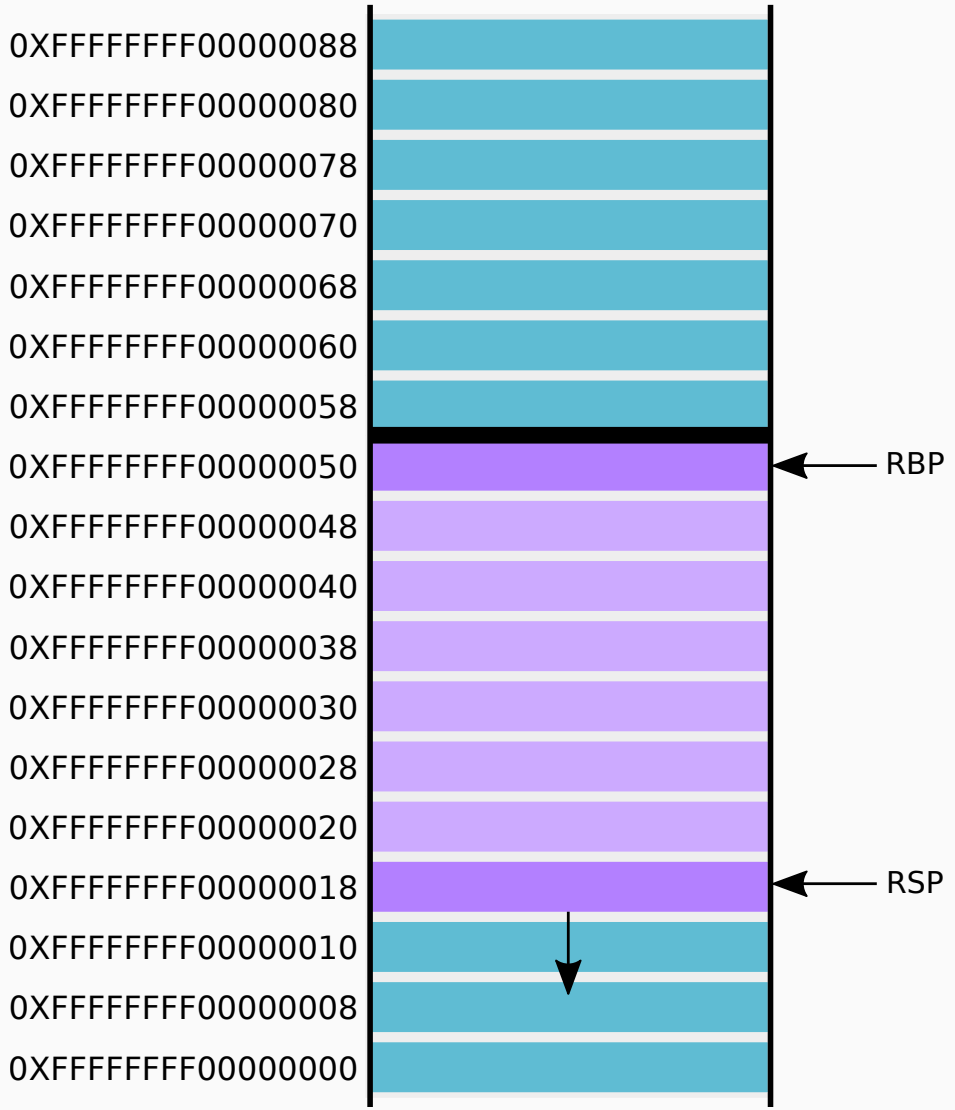
Registros

- ESP
- EBP

Instrucciones

- PUSH
- POP

Uso de la pila



Alineación

- 8 bytes (en 64 bits)
- 16 bytes para llamar funciones de C

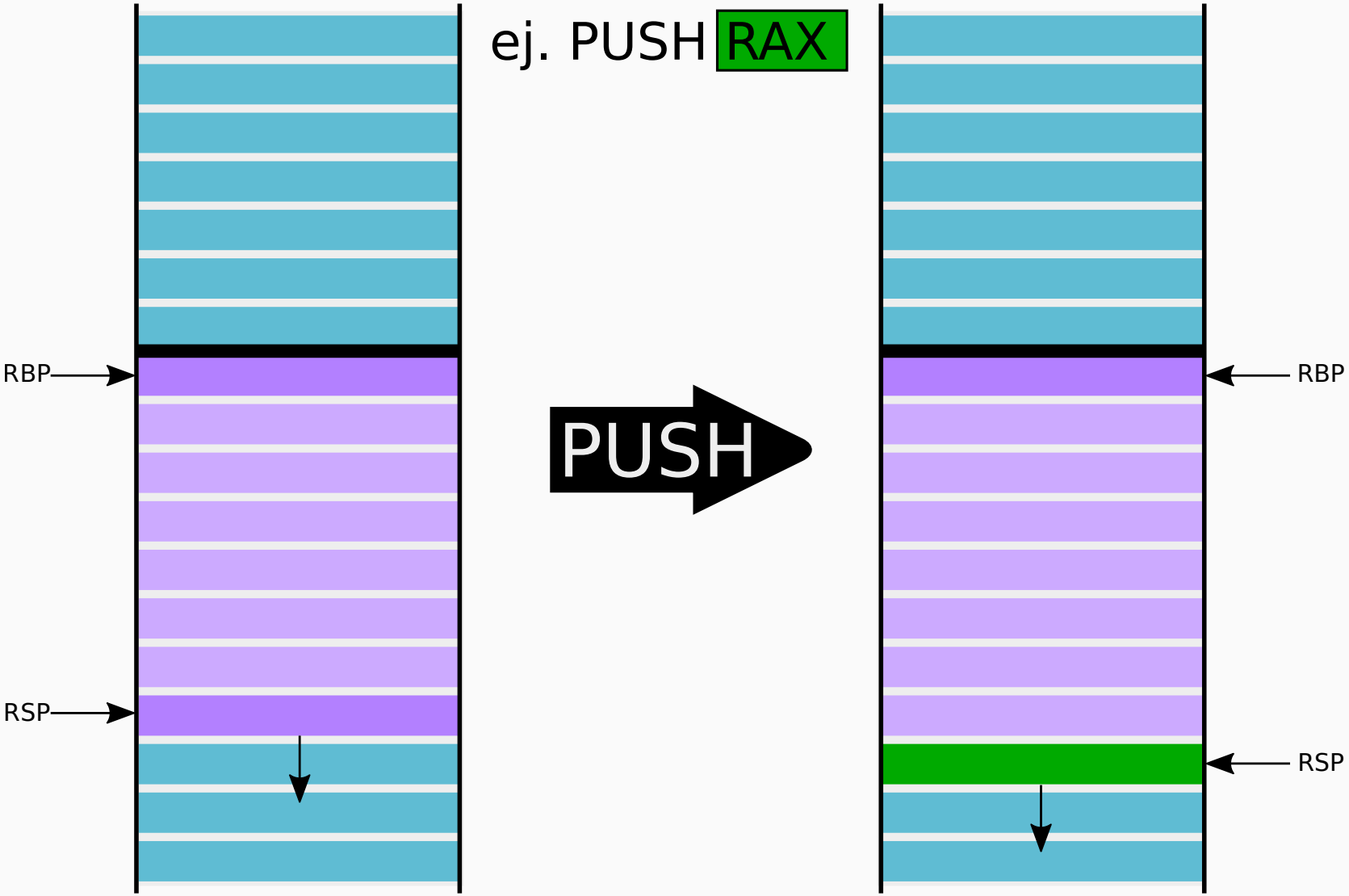
Registros

- RSP
- RBP

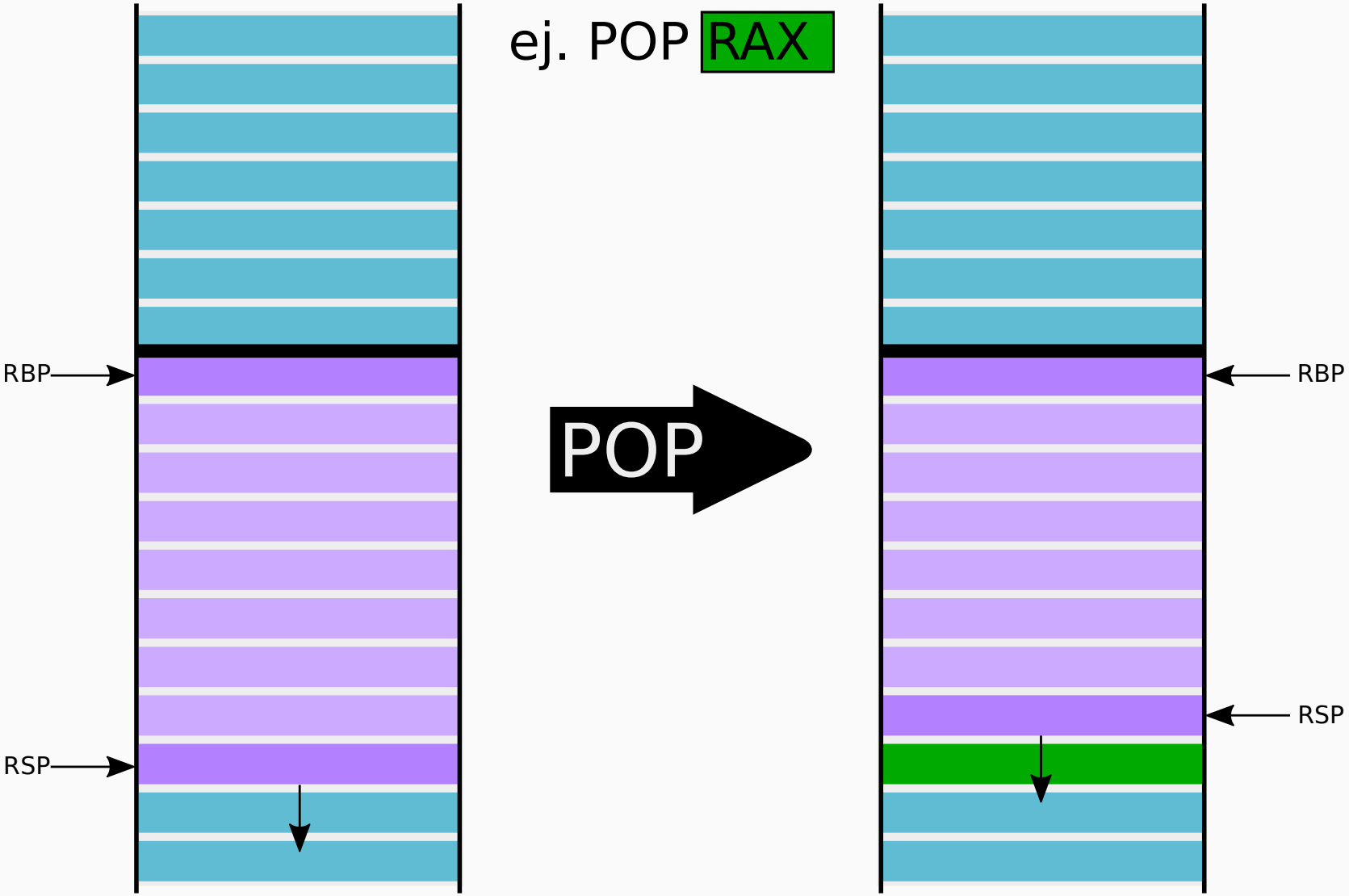
Instrucciones

- PUSH
- POP

Uso de la pila



Uso de la pila



System V - Versión de 64 bits

En 64 bits `System V` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son **no volátiles**
- El valor de retorno será almacenado en `RAX` para valores enteros[1] y en `XMM0` para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

[1] Para el ABI los punteros son enteros ¯_(ツ)_/¯

System V - Versión de 64 bits

En 64 bits `System V` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son **no volátiles**
- El valor de retorno será almacenado en `RAX` para valores enteros[1] y en `XMM0` para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

[1] Para el ABI los punteros son enteros ¯_(ツ)_/¯

System V - Versión de 64 bits

En 64 bits `System V` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son **no volátiles**
- El valor de retorno será almacenado en `RAX` para valores enteros[1] y en `XMM0` para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

[1] Para el ABI los punteros son enteros ¯_(ツ)_/¯

System V - Versión de 64 bits

En 64 bits `System V` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son **no volátiles**
- El valor de retorno será almacenado en `RAX` para valores enteros[1] y en `XMM0` para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

[1] Para el ABI los punteros son enteros ¯_(ツ)_/¯

System V - Versión de 64 bits

En 64 bits `System V` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son **no volátiles**
- El valor de retorno será almacenado en `RAX` para valores enteros[1] y en `XMM0` para flotantes
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`)(¿RSP volátil?)
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes

[1] Para el ABI los punteros son enteros ¯_(ツ)_/¯

System V - Versión de 64 bits

Aclaración: Donde diga **de derecha a izquierda** o **de izquierda a derecha** debemos entender que nos referimos al orden de los parámetros en la declaración de la función en el encabezado `.h`.

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits

En 64 bits la `System V` define lo siguiente:

- Los parámetros enteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente
- Los parámetros flotantes se pasan de izquierda a derecha en `XMM0`, `XMM1`, `XMM2`, `XMM3`, `XMM4`, `XMM5`, `XMM6`, `XMM7` respectivamente
- Si no hay registros disponibles para los parámetros enteros y/o flotantes se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH`.
- Las estructuras se tratan de una forma especial (ver referencia). Si son grandes se pasa un puntero a la misma como parámetro.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 64 bits (machete)

Para que tengan a mano en la **primera parte de la materia (64 bits)**:

No volátiles	`RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` (¿`RSP`?)
Valor de retorno	`RAX`enteros/punteros, `XMM0` flotantes
Entero, puntero	`RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` (izq. a der.)
Flotantes	`XMM0`, `XMM1`, . . . , `XMM7` (izq. a der.)
¿No hay registros?	`PUSH` a la pila (der. a izq.)
Inv. de pila	Todo `PUSH`/`SUB` debe tener su `POP`/`ADD`
Llamada a func.	Pila alineada a 16 bytes

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits

En 32 bits la `System V` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son **no volátiles**.
- El valor de retorno será almacenado en `EAX`.
- Al salir de la función llamada la pila debe encontrarse en el mismo estado en el que estaba al ingresar (todo `PUSH` debe tener su `POP`).
- Los parámetros se pasarán de **derecha a izquierda** a través de la pila haciendo `PUSH` (¿ESP volátil?.)
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.
- Si el valor a retornar es una estructura grande la función llamadora reserva espacio (parámetro implícito).

System V - Versión de 32 bits (machete)

Para que tengan a mano en la **segunda parte de la materia (32 bits)**:

No volátiles	`EBP`, `EBX`, `ESI` y `EDI` (¿`ESP`?)
Valor de retorno	`EAX`
Parámetros	`PUSH` a la pila (der. a izq.)
Inv. de pila	Todo `PUSH`/`SUB` debe tener su `POP`/`ADD`
Llamada a func	Pila alineada a 16 bytes

Pregunta: ¿Por qué hace falta alinear la pila a 16 bytes si hacemos una llamada a otra biblioteca?

Pregunta: ¿Por qué hace falta alinear la pila a 16 bytes si hacemos una llamada a otra biblioteca?

Respuesta: Las funciones pueden hacer uso de operaciones de registros largos (`XMM`, `YMM`) que requieren datos alineados a 16 bytes, es por esto que el contrato de uso de un conjunto de instrucciones del procesador se traduce en un contrato de uso de nuestras funciones de bajo nivel.