Práctica de Organización del Computador II

Convención C

Segundo Cuatrimestre 2022

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.

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)

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.

¿Qué es una declaración de función en C?

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

¿Qué es una declaración de función en C?

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

- Declara la existencia de una función llamada `product`.

¿Qué es una declaración de función en C?

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

- Declara la existencia de una función llamada `product`.
- Indica que una llamada o implementación de una función llamada `product` devuelve un parámetro de tipo `int32_t`.

¿Qué es una declaración de función en C?

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

- Declara la existencia de una función llamada `product`.
- Indica que una llamada o implementación de una función llamada `product` devuelve un parámetro de tipo `int32_t`.
- Indica que una llamada o implementación de una función llamada `product` toma un parámetro de tipo `int32_t*` y otro de tipo `uin32_t`.

¿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).

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 en C que comparta el nombre de la función debe compartir el tipo devuelto y cantidad y tipo de parámetros.

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 en C que comparta el nombre de la función debe compartir el tipo devuelto y 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 arquitectura 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 arquitectura particular.

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

¿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?

¿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: Correcto, para eso vamos a tener que familiarizarnos con 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 arquitectura.

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 arquitectura.
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

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

Interfaz binaria de aplicación

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

- Formato de archivos objeto y ejecutables

Interfaz binaria de aplicación

Una ABI completa 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

Interfaz binaria de aplicación

Una ABI completa 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

Una ABI completa 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

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

- El set de instrucciones.

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

- El set de instrucciones.
- Los tipos de datos primitivos.

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

- El set de instrucciones.
- Los tipos de datos primitivos.
- La forma de realizar y pasar información a funciones de sistema (Sys Calls).

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

- El set de instrucciones.
- Los tipos de datos primitivos.
- La forma de realizar y pasar información a funciones de sistema (Sys Calls).

Que ya lo hemos visto, pero también sobre cosas que todavía no ejercitamos:

Una ABI parcial, que es lo que sí vamos a ver, va a definir contratos sobre:

- El set de instrucciones.
- Los tipos de datos primitivos.
- La forma de realizar y pasar información a funciones de sistema (Sys Calls).

Que ya lo hemos visto, pero también sobre cosas que todavía no ejercitamos:

- La forma de realizar y pasar información entre funciones de usuarix (Convención C).

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

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

- ¿A través de los registros de propósito general?

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

- ¿A través de los registros de propósito general?
- ¿A través de la pila?

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

- ¿A través de los registros de propósito general?
- ¿A través de la pila?

Respuesta:Vamos a utilizar la pila y los registros de propósito general.

Pregunta:¿Qué pasa con los registros que están siendo usados por la función cuando se realiza una llamada? ¿Van a conservar sus valores al regresar?

Respuesta:Vamos a definir un conjunto de registros no volátiles que deben preservarse. Su valor puede cambiar durante la ejecución de la función llamada pero deben restaurarse antes de regresar a la llamadora (invariante de función).

Respuesta:Vamos a definir un conjunto de registros no volátiles que deben preservarse. Su valor puede cambiar durante la ejecución de la función llamada pero deben restaurarse antes de regresar a la llamadora (invariante de función).
El resto de los registros serán volátiles y la función llamada no tiene obligación de restaurarlos antes de terminar su ejecución.

Convención C

La convención C define dos contratos de función:

La convención C define dos contratos de función:

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

Convención C

La convención C define dos contratos de función:

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

La convención C define dos contratos de función:

- 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`.

Convención C

La convención C define dos contratos de función:

- 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`.

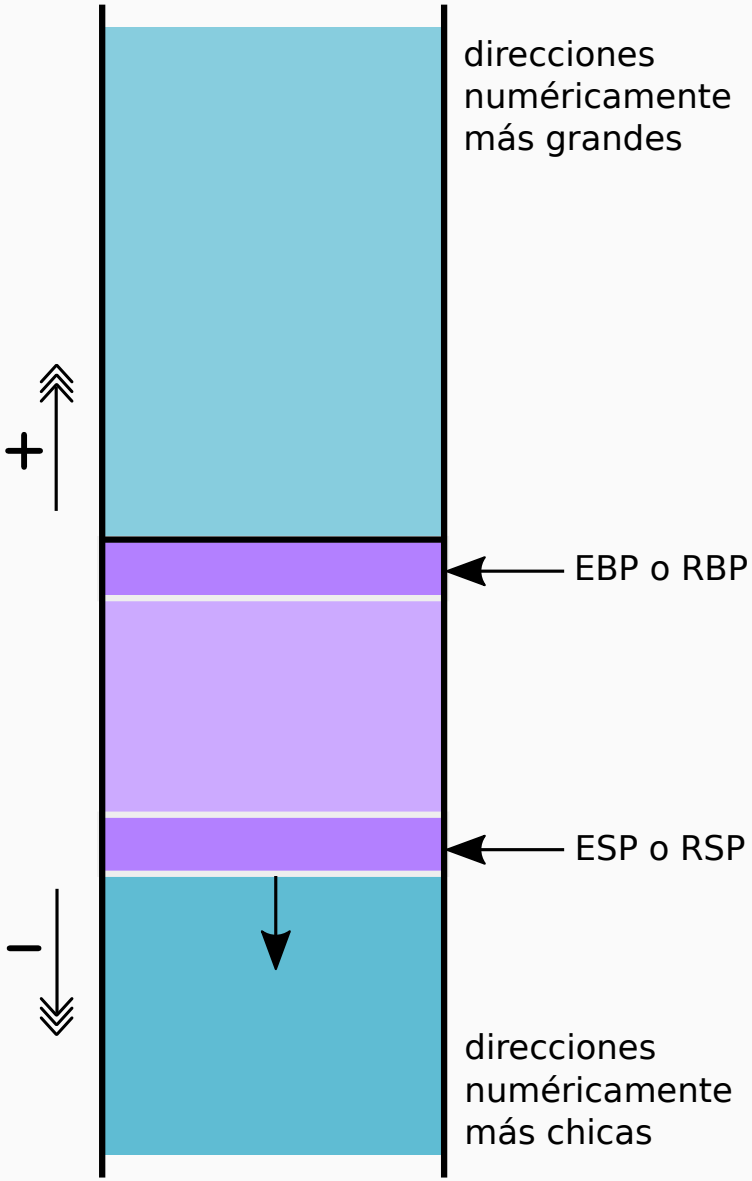
El primero se va a usar en la primera parte de la materia (programación de aplicaciones) y el segundo en la segunda parte (programación de sistema).

Uso de la pila

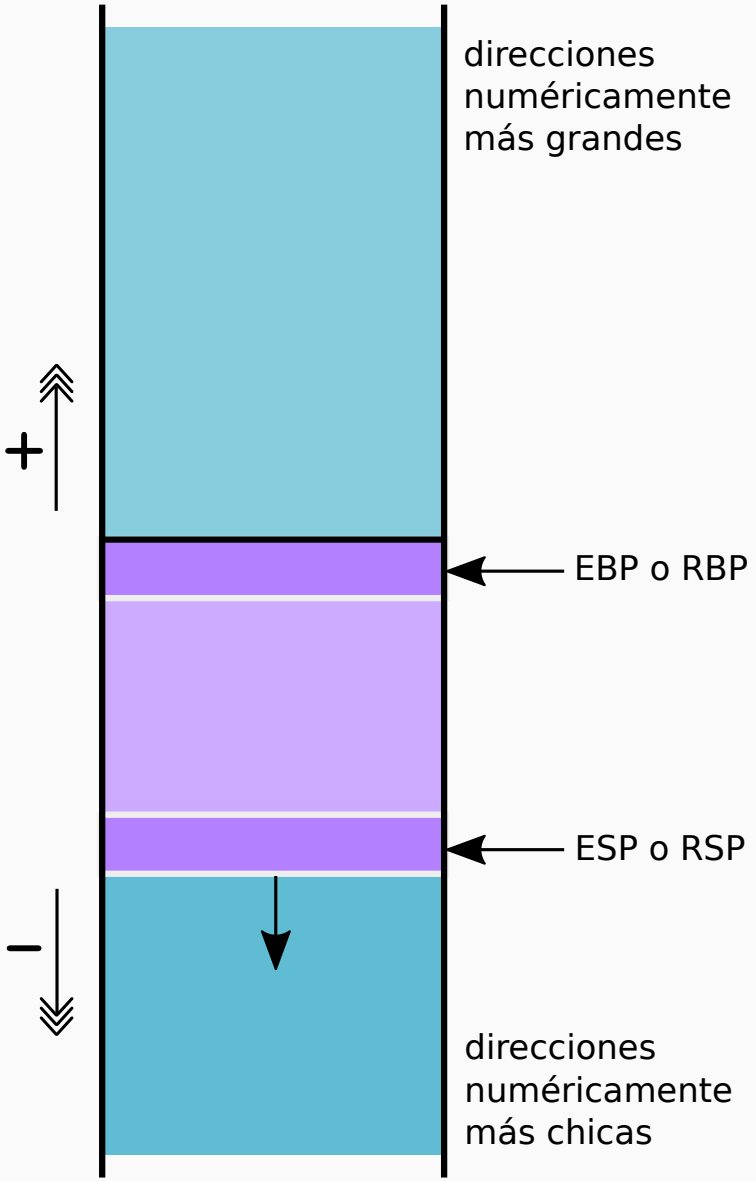
La pila

- 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



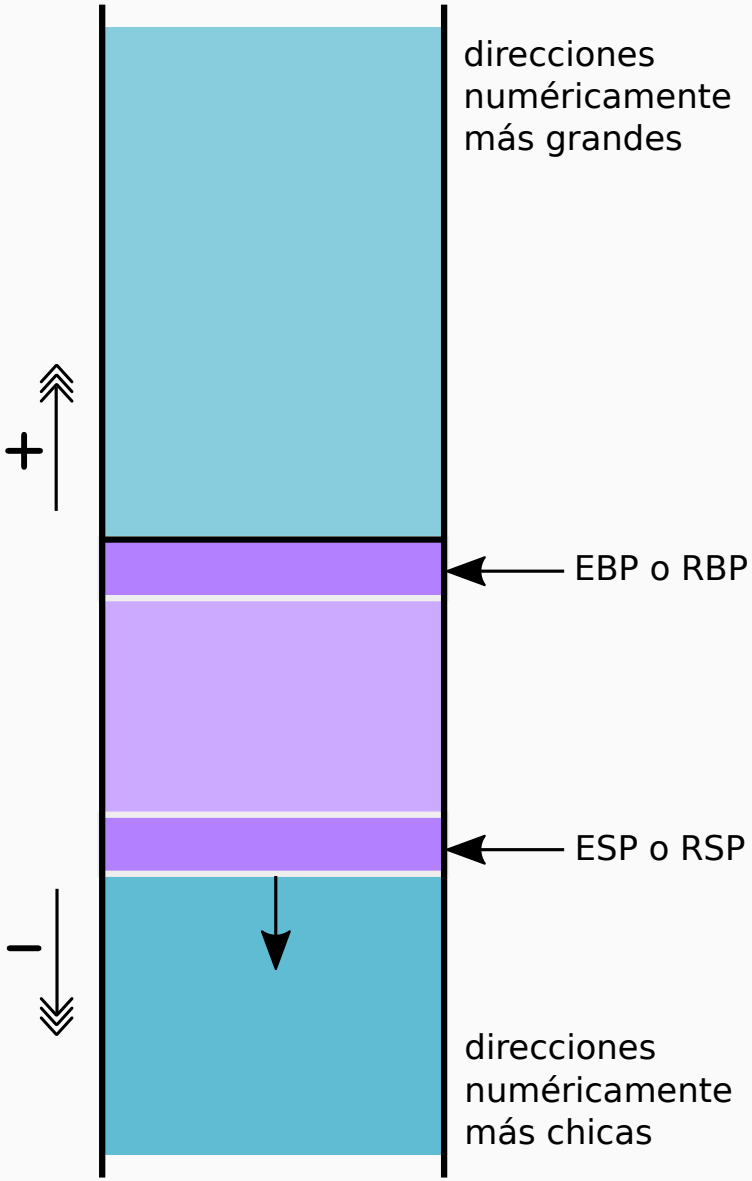
Uso de la pila



En 32 bits

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

Uso de la pila



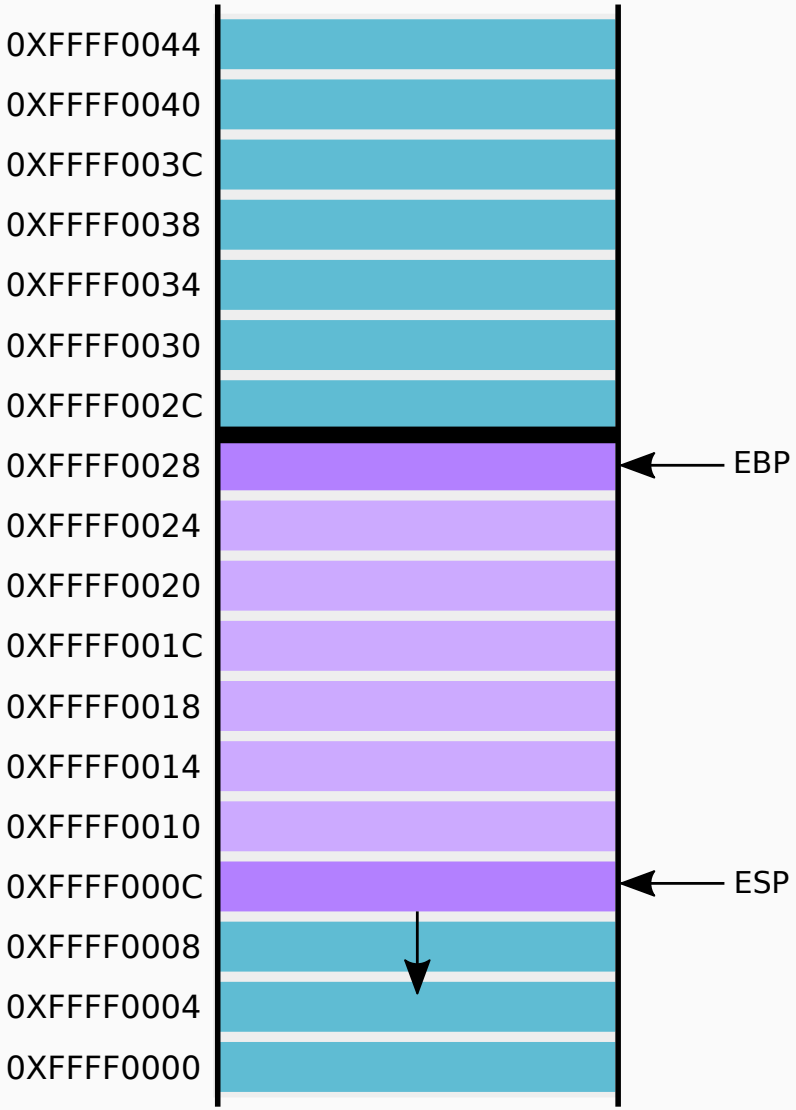
En 32 bits

- 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)

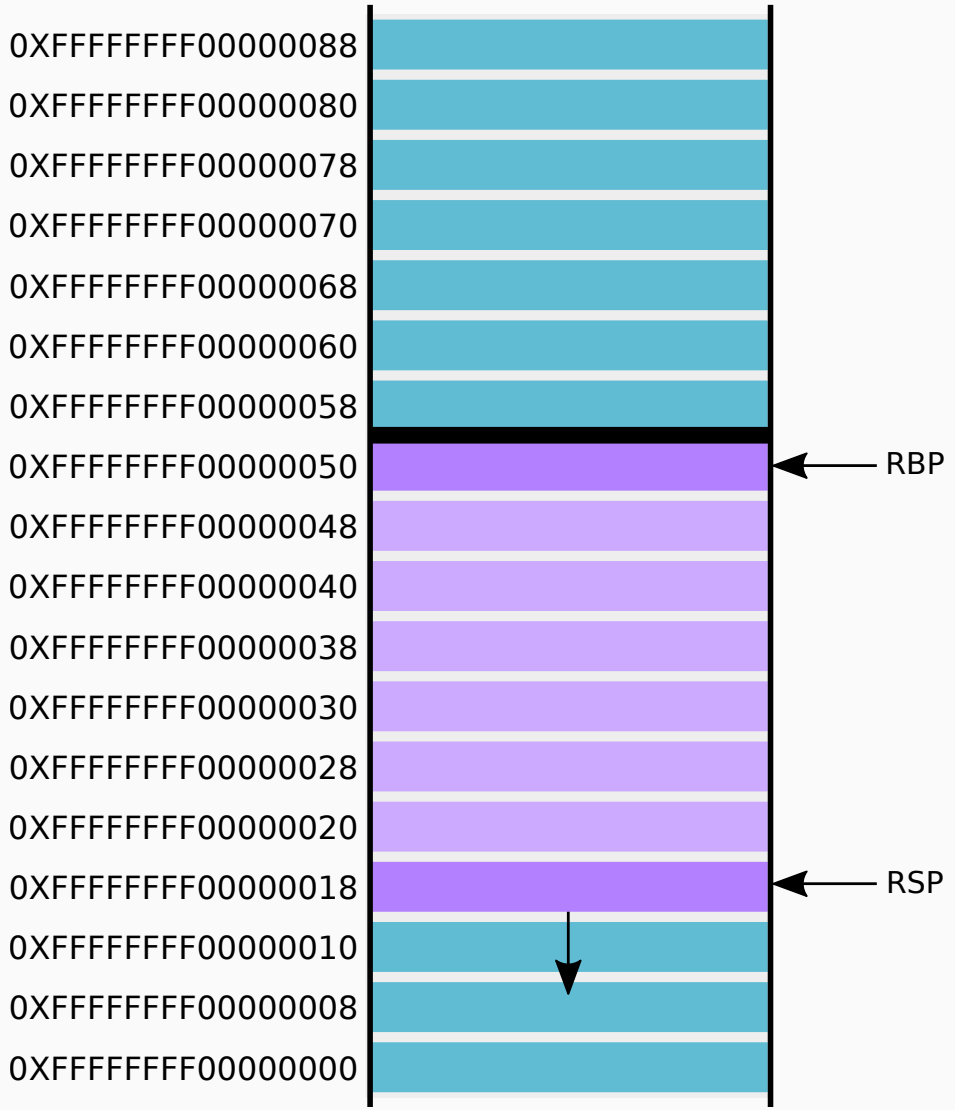
Registros

- ESP
- EBP

Instrucciones

- PUSH
- POP

Uso de la pila



Alineación

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

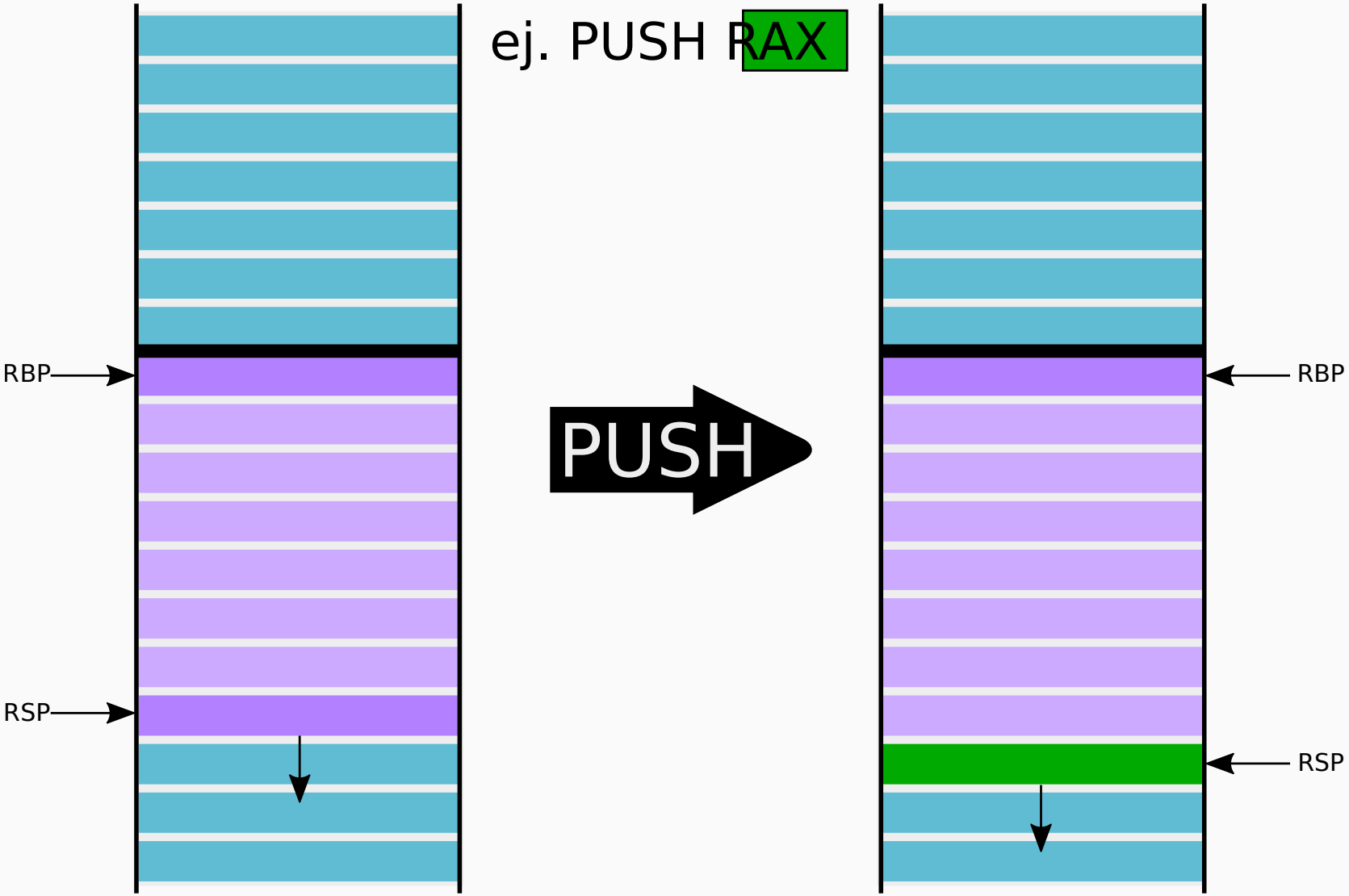
Registros

- RSP
- RBP

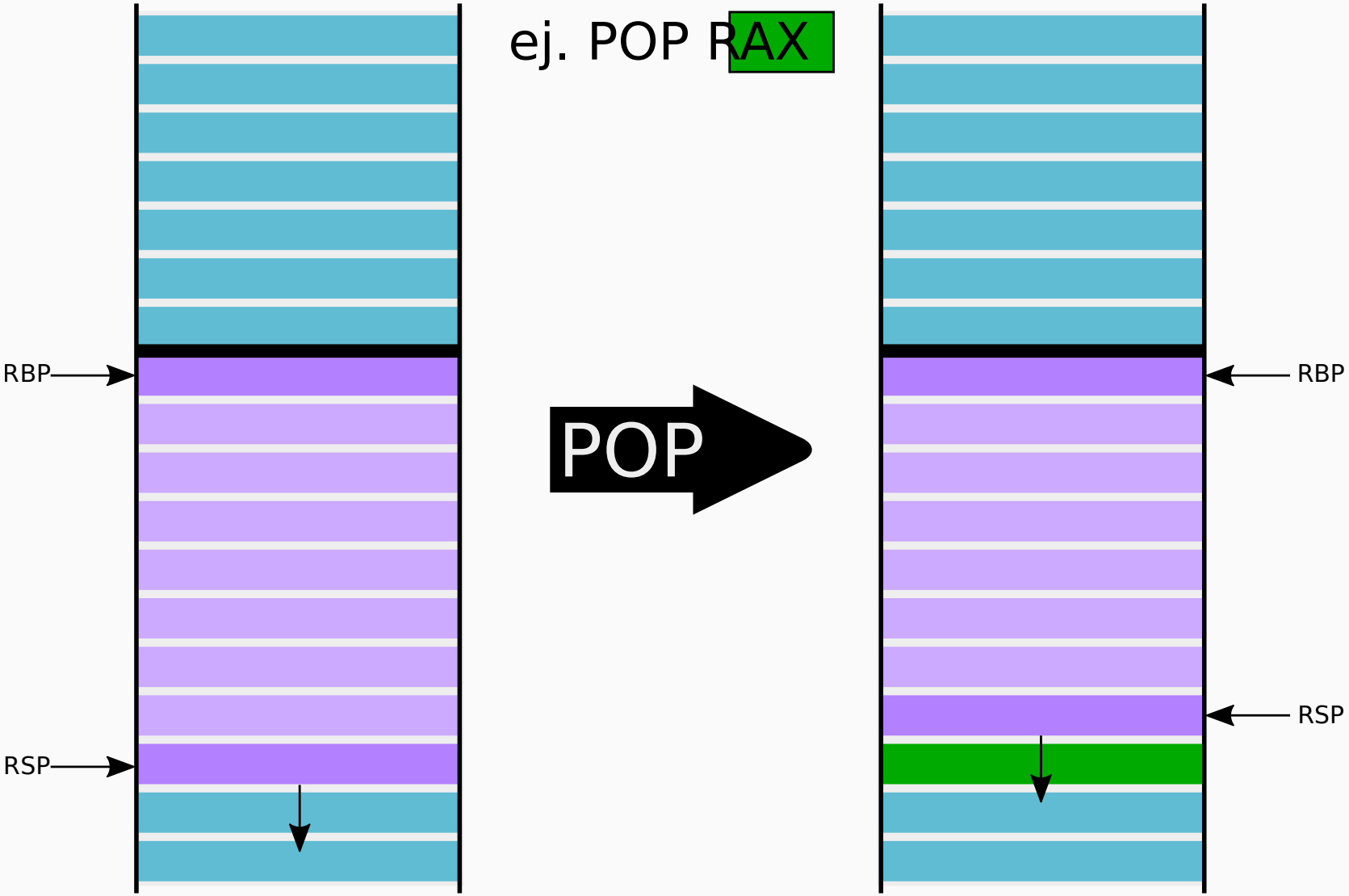
Instrucciones

- PUSH
- POP

Uso de la pila



Uso de la pila



Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los registros `RBX`, `RBP`, `R12`, `R13`, `R14` y `R15` son no volátiles.

Convención C - Versión de 64 bits

En 64 bits la `convención C` 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 y punteros y en `XMM0` para flotantes.

En 64 bits la `convención C` 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 y punteros 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`).

En 64 bits la `convención C` 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 y punteros 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`).
- Antes de realizar una llamada a una función, la pila debe estar alineada a 16 bytes.

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`.

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los parámetros enteros y los punteros se pasan de izquierda a derecha en `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9` respectivamente.

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los parámetros enteros y los punteros 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.

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los parámetros enteros y los punteros 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`.

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los parámetros enteros y los punteros 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.

Convención C - Versión de 64 bits

En 64 bits la `convención C` define lo siguiente:

- Los parámetros enteros y los punteros 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).

Convención C - Versión de 64 bits (machete)

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

Convención C - 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`
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 func. `C`:	pila alineada a 16 bytes

Convención C - Versión de 32 bits

En 32 bits la `convención C` define lo siguiente:

Convención C - Versión de 32 bits

En 32 bits la `convención C` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son no volátiles.

Convención C - Versión de 32 bits

En 32 bits la `convención C` define lo siguiente:

- Los registros `EBX`, `EBP`, `ESI` y `EDI` son no volátiles.
- El valor de retorno será almacenado en `EAX`.

Convención C - Versión de 32 bits

En 32 bits la `convención C` 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`).

Convención C - Versión de 32 bits

En 32 bits la `convención C` 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`.

Convención C - Versión de 32 bits

En 32 bits la `convención C` 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`.
- Antes de realizar una llamada a una función la pila debe quedar alineada a 16 bytes.

Convención C - Versión de 32 bits

En 32 bits la `convención C` 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`.
- 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).

Convención C - Versión de 32 bits (machete)

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

Convención C - 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`
Valor de retorno:	`EAX`
Parámetros:	`PUSH` a la pila(der. a izq.)
Inv. de pila:	Todo `PUSH`/`SUB` debe tener su `POP`/`ADD`
Llamada func. `C`:	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.