Arquitectura de Ordenadores

Arquitectura del Procesador Intel Pentium

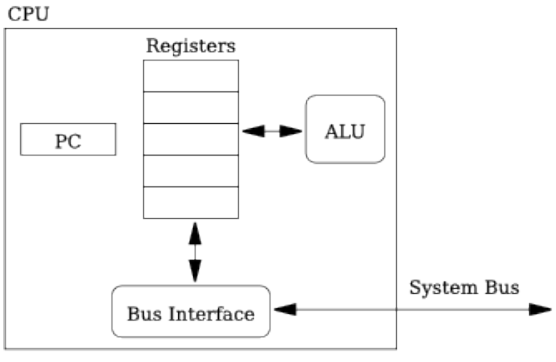
Abelardo Pardo
abel@it.uc3m.es

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Entorno de Ejecución del Procesador Intel Pentium

- ¿Qué **componentes internos** tiene el procesador?

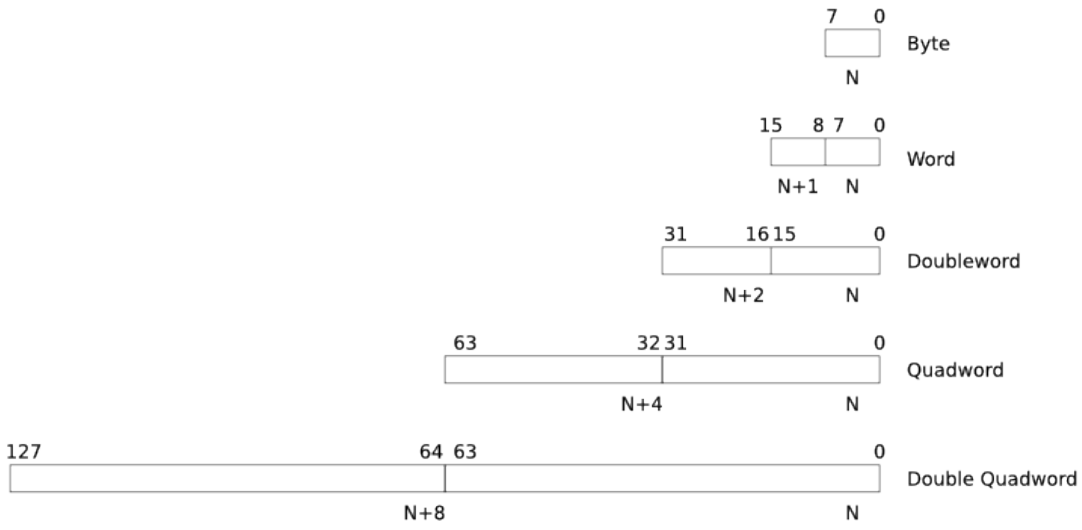
- ¿Cómo se comunica **con los bloques externos**?



El Espacio de Direcciones

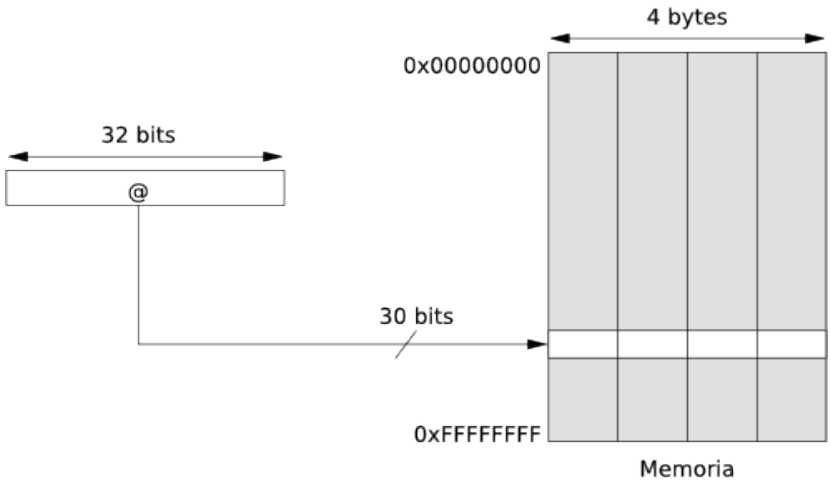
- El conjunto de **posibles direcciones de memoria** se denomina **espacio de direcciones**.
- Un procesador define su **espacio de direcciones** mediante el número de bits que utiliza para codificar una dirección de memoria.
- El pentium tiene **dos modos** de gestión de memoria: **modo plano** y **modo segmentado**.
- Se estudiará únicamente el **modo plano**.
- La dirección de memoria se codifica con **32 bits**.
- El procesador **en modo plano** puede manejar **como máximo** 4 gigabytes de memoria.

Unidades Fundamentales de Datos

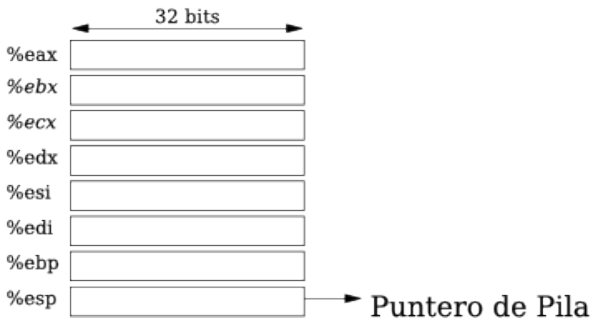


Transferencia de Datos de/a Memoria

- Para aumentar la **velocidad de transmisión** de datos con la memoria el procesador transfiere siempre **4 bytes**.
- La dirección de memoria que transporta el bus es siempre **un múltiplo de 4** (o sea, $32 \% 2 = 0$).
- **Ejemplo:** Si se envia la dirección **0x00000033** la memoria devuelve los bytes en las posiciones **0x30** a **0x33**.
- Este esquema de acceso permite **ignorar los dos bits menos significativos de la dirección**.
- Los 2 bits menos significativos los utiliza el procesador para **seleccionar el byte pertinente**.



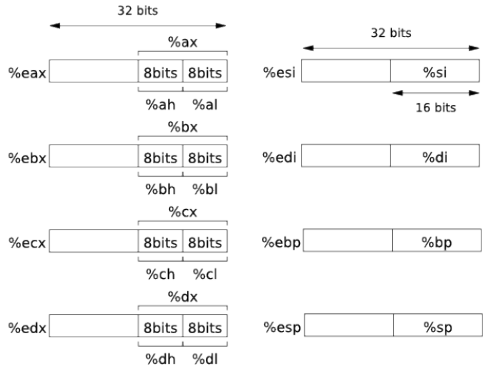
Registros de Propósito General



- Ocho registros de **32 bits**
- Se comportan como **celdas de memoria**.
- Permiten la **lectura y escritura** de datos.
- Se utilizan para almacenar los **datos temporales** utilizados por las instrucciones máquina.

Porciones de los Registros Generales

- Se pueden manipular **porciones de los registros generales**.
- Se pueden manipular los **dos bytes menos significativos** de los registros `%eax, %ebx, %ecx` y `%edx`.
- Su nombre se obtiene **eliminando el prefijo 'E'** y **sustituyendo la 'X'** por **'H'** o **'L'** para acceder al byte más o menos significativo respectivamente.
- Se puede acceder a los **16 bits menos significativos** de cada registro.
- El nombre de la porción se obtiene **eliminando la letra 'E'**.



- **Ejemplos:**
  - `MOV $5, %eax`. Se mueven 32 bits.
  - `MOV $5, %al`. Se mueven 8 bits.

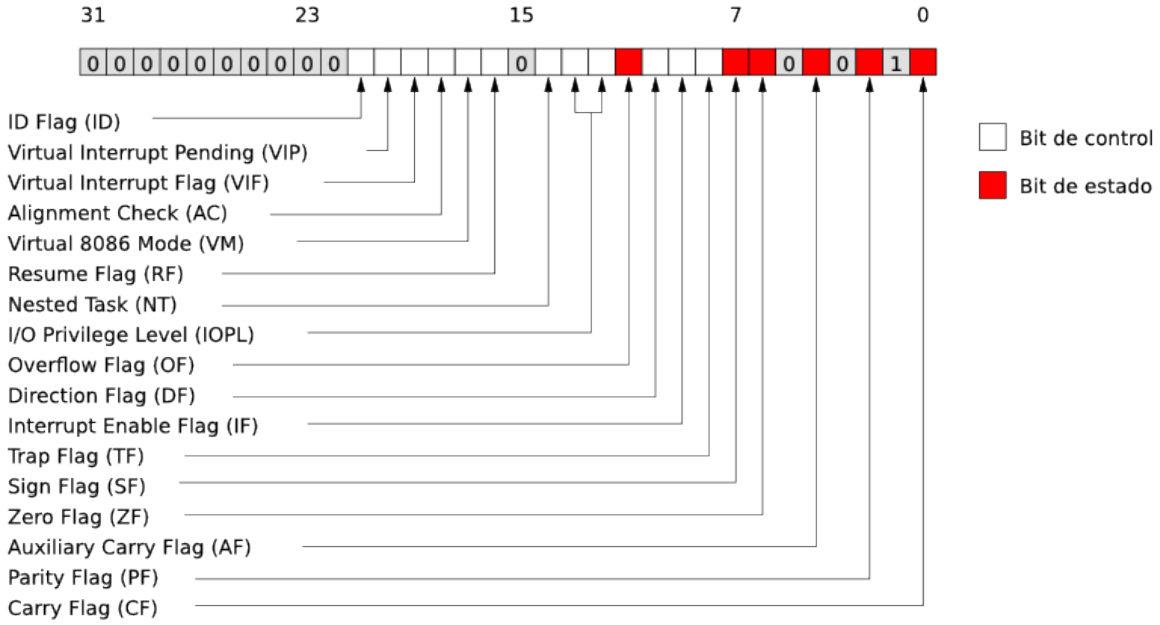
Registro de Estado y Control de Programa

- El procesador almacena en el **registro de estado y control** un conjunto de bits de 3 tipos:

1. **Condición:** Reflejan una condición actual de procesador. Sólo se pueden leer.
2. **Control:** El procesador se comporta de forma diferente dependiendo de los valores. Se permiten leer y escribir.
3. **Sistema:** Valores que sólo se pueden modificar con instrucciones especiales.

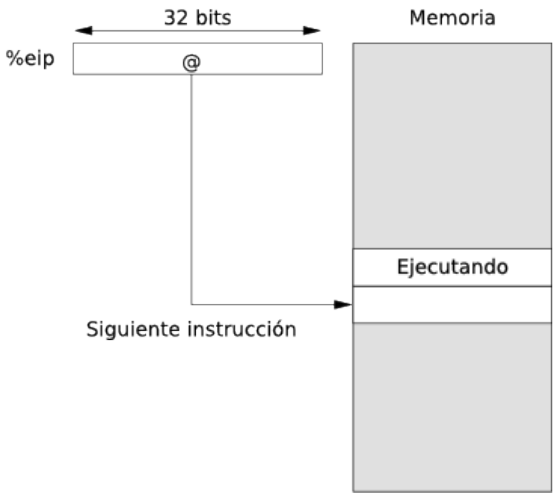
Registro de Estado y Control de Programa más importantes

- De los bits de este registro se destacan **los siguientes**:
  1. **Bit de Acarreo (CF):** Su valor es **el acarreo** de la última operación.
  2. **Bit de Cero (ZF):** Su valor es **1** si la última operación realizada ha sido **cero**.
  3. **Bit de Signo (SF):** Contiene el signo del resultado de la última operación.
  4. **Bit de Desbordamiento (OF):** Su valor es **1** si la última operación ha producido **overflow**.



El Registro Contador de Programa

- El **único** cometido del procesador es la ejecución de instrucciones.
- En todo momento se debe saber **en qué posición de memoria** está almacenada la siguiente instrucción a ejecutar.
- Este valor se almacena en un registro conocido **genéricamente** como **contador de programa** (PC: program counter).
- La **próxima instrucción a ejecutar** no tiene por qué estar almacenada en la siguiente posición de memoria.
- El pentium tiene un **contador de programa de 32 bits**.
- Este registro no es modificable **directamente**.
- Sólo **ciertas instrucciones** modifican su contenido.
- En los manuales del procesador se refieren a él como **Instrucion Pointer** o **EIP**

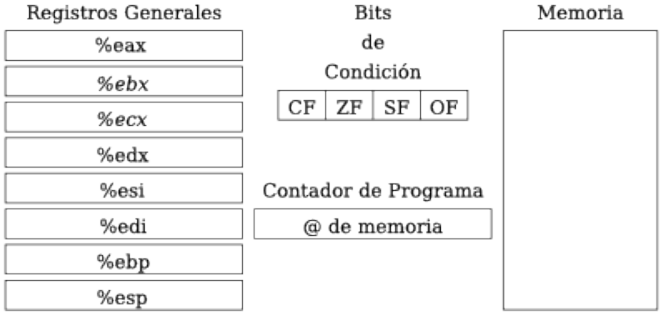


Otros Registros del Pentium

- Aparte de los registros descritos, el Pentium cuenta con registros adicionales que **por simplicidad** no se van a considerar.
  - **6 registros de 16 bits** para manipular la memoria en modo segmentado.
  - **8 registros de 80 bits** para operaciones con números en coma flotante.
  - **3 registros de 16 bits** para almacenar información de control y estado adicional del procesador.
  - **2 registros de 48 bits** para almacenar la dirección y datos de la siguiente instrucción de coma flotante.
  - **8 registros de 64 bits** para instrucciones SIMD o instrucciones que se ejecutan sobre múltiples datos (single instruction multiple data)
  - **8 registros de 128 bits** para instrucciones SIMD sobre números de coma flotante.
  - Registros para gestión de memoria, detección de errores, depuración de programas, control de rendimiento, etc.

Estado Visible de un Programa

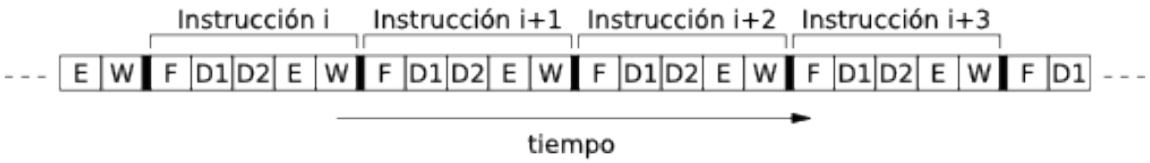
- De todos los registros del procesador **se considerarán únicamente** los registros de propósito general.
- El **estado de un programa** se define como el conjunto de valores de los que depende su ejecución.
- Se trabajará con una **versión simplificada** del procesador en la que el estado de un programa está contenido en:
  - **Registros de propósito general**
  - **Bits CF, ZF, SF y OF**
  - **Contador de programa**
  - **Contenido de Memoria**



Ciclo de Ejecución

- Desde el momento en que se enciende el ordenador el procesador **comienza a ejecutar instrucciones**.
- La ejecución de instrucciones se divide en **fases**.
- Estas fases son **diferentes** dependiendo del tipo de instrucción, pero asumiremos que **todas ellas siguen un patron similar**.
- La ejecución genérica de una instrucción consta de **cinco fases**:

1. Carga de Instrucción (Fetch - F)
2. Decodificación (Decode - D)
3. Ejecución (Execute - E)
4. Acceso a Memoria (Memory - M)
5. Escritura de resultado y actualización del contador de programa (Write Back - WB)

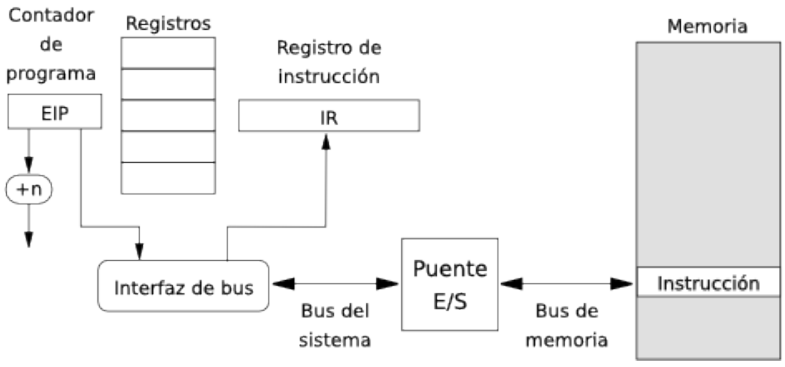


Carga de Instrucción (Fetch)

- Operaciones realizadas en la fase de **Fetch**:

1. Se leen de memoria los 4 bytes contenidos en la posición **apuntada por el contador de programa**.
2. Se interpreta la información obtenida y se decide **qué tipo de instrucción a ejecutar**.
3. Si es preciso se lee el resto de la instrucción y se extraen los operandos **contenidos en la propia instrucción**. Para ello se obtienen los bytes adicionales de memoria que sea preciso.
4. Se obtienen (si así está codificado en la instrucción) **los registros especificados como operandos**.
5. Calcula el valor del contador de programa como la posición **siguiente** a la última cargada en memoria.

Carga de Instrucción (Fetch) II

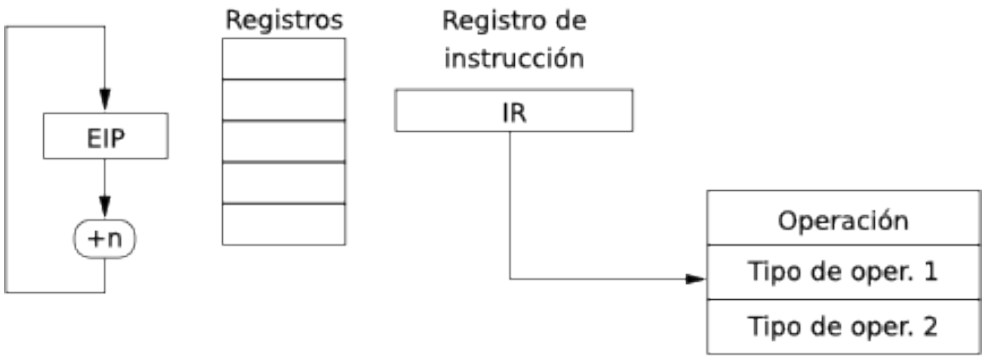


- El nuevo valor del PC **se calcula pero no se actualiza**.

Decodificación (Decode)

- Operaciones realizadas en la fase de **Decode**:

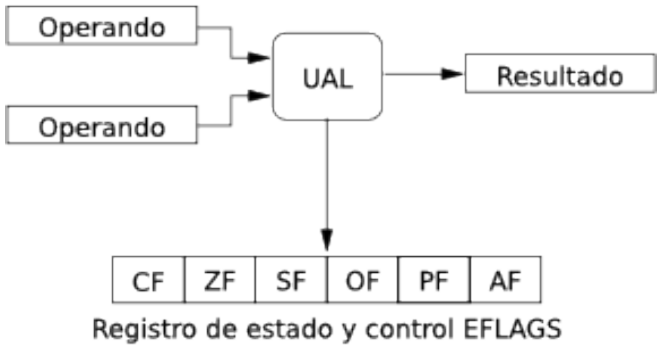
1. Se leen los valores de **los registros especificados como operandos**.
2. Si es preciso **se accede a memoria** para obtener operandos adicionales.



Ejecución (Execute)

- Operaciones realizadas en la fase de **Execute**:

1. La **unidad aritmético-lógica** (ALU) realiza la operación pertinente.
2. Se actualizan los valores del **registro de estado y control** del procesador.
3. Se almacena **el resultado de la operación** en un registro interno.



Acceso a Memoria (Memory)

- Operaciones realizadas en la fase de **Memory**:

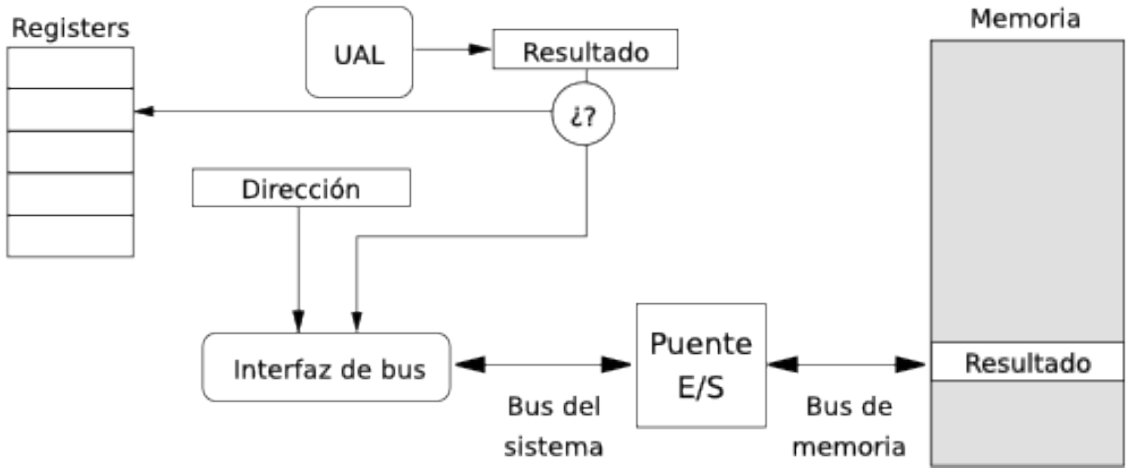
  - Se **lee un dato** de una determinada posición de memoria y se almacena en un registro interno.
  - Se **escribe un dato** en una determinada posición de memoria.
  - Dependiendo del **tipo de instrucción** se realiza una operación de lectura, escritura o **ninguna**.

Escritura de Resultado (Write Back)

- Operaciones realizadas en la fase de **Write Back**:

1. Escribe **hasta dos resultados** simultáneamente en los registros de propósito general.

2. Actualiza el registro **contador de programa** con la dirección de la siguiente instrucción a ejecutar.



Ejemplo de Ejecución de una Instrucción Máquina

- La instrucción **MOV $128, %eax** carga el valor decimal **128** en el registro %eax.

- En el Pentium se codifica como **0xB880000000**.

1. **Fetch:** Se carga la instrucción de memoria, se detecta que es una instrucción **MOV**. Se extrae el número $128 y la referencia al registro %eax. Se calcula la suma del contador de programa y 5 (tamaño de la instrucción en ejecución)

2. **Decode:** No se precisa ningún valor de los registros como operando, por lo que en esta fase de esta instrucción no se hace nada.

3. **Execute:** El valor $128 se suma al valor 0. Se almacen el resultado en un registro interno y, en el caso de instrucciones aritméticas, se modifican los bits de estado.

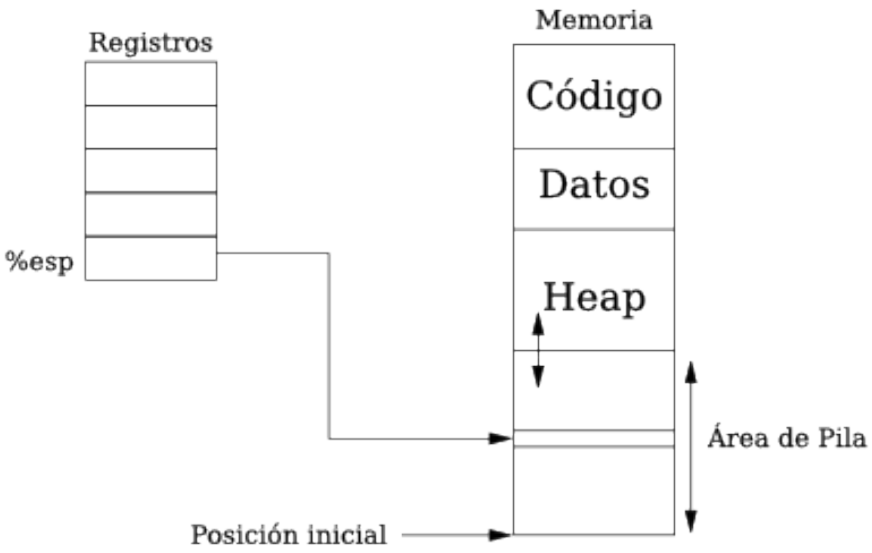
4. **Memory:** No se transfieren datos entre el procesador y memoria.

5. **Write Back:** Se almacena el resultado en el registro %eax y se actualiza el contador de programa con el valor anterior mas 5 (tamaño de la instrucción ejecutada).

La Pila (Stack)

- No todos los registros de propósito general contienen **datos arbitrarios**.
- El registro **%esp** contiene **una dirección de una porción de memoria** denominada **la pila** (Stack).
- Al registro %esp también se le conoce como **puntero de pila** o **stack pointer (SP)**.
- El procesador ejecuta instrucciones que **asumen** que el registro %esp contiene este tipo de información.
- Las principales operaciones que utilizan el registro %esp son: `push`, `pop`, `call` y `ret`
- El efecto es **similar** a depositar y extraer contenido de un conjunto de datos apilados.
- A pesar de esta función, el registro %esp **puede ser modificado por cualquier instrucción**.
- La porción de memoria que ocupa la pila no tiene nada especial y puede ser **manipulada por cualquier instrucción**.

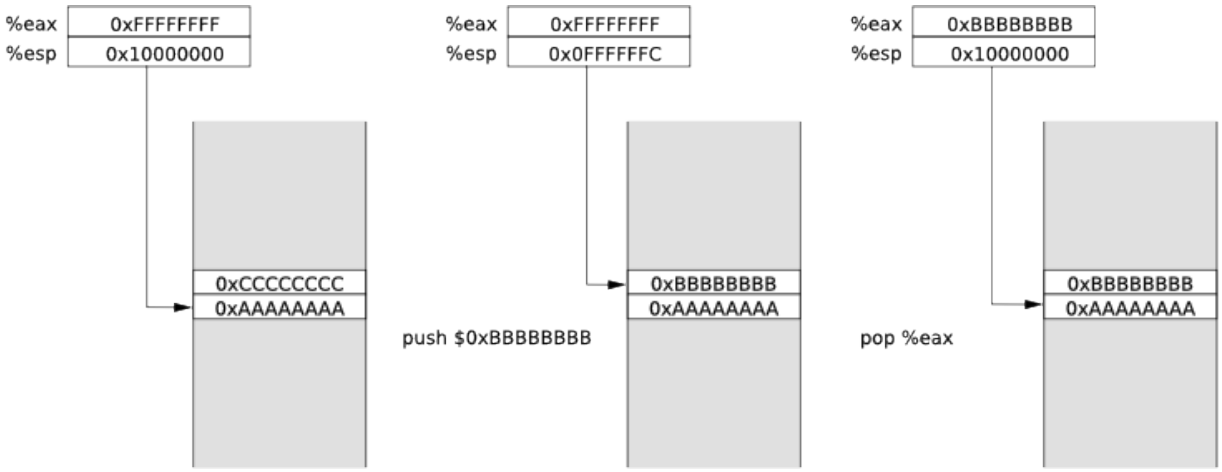
La Pila (Stack) II



- Los límites de la pila **los crea el sistema operativo para cada programa**.
- Si se ejecuta una **instrucción de pila** (`push`, `pop`) que salga de estos límites se produce un error en el programa.
- La **posición inicial** a la que apunta %esp es **la dirección de memoria más alta de la pila** o **cima de la pila**.

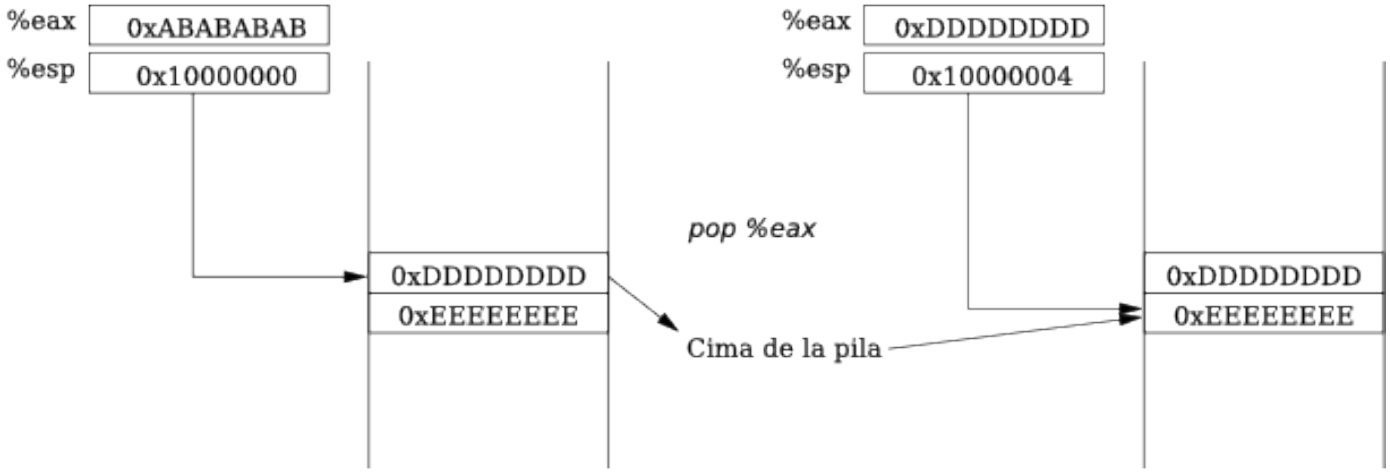
La Intrucción Push

- **Formato:** `push <operando>`
- El operando puede ser de **16 o 32** bits. Se asumirá siempre operandos de **32** bits.
- **Efecto:** Se resta **4 a la dirección contenida en %esp** y **se almacena el operando en dicha posición.**
- El dato que estaba previamente en esa posición de memoria **se pierde**.



La Instrucción Pop

- **Formato:** `pop <lugar>`
- **Efecto:** Se transfiere el dato de 32 bits al lugar especificado en la dirección y se añade 4 a la dirección contenida en %esp.
- Es una operación de lectura de memoria por lo que **el dato permanece en memoria**.
- El dato que se pierde es el que había previamente **en el lugar especificado como operando.**



Preguntas Sobre la Pila

- ¿Cuántos registros cambian de contenido tras ejecutar la instrucción `push %eax`?
- ¿Cuántos registros (como máximo) cambian de contenido tras ejecutar la instrucción `pop %eax`?
- ¿Cuántos registros contienen datos diferentes al ejecutar la instrucción `push %eax` seguida de la instrucción `pop %eax`?
- ¿Cuántas posiciones de memoria han modificado su valor (como máximo) tras ejecutar **las dos instrucciones de la pregunta anterior**?
- La instrucción `pop %edx` no ha modificado el contenido del registro %edx ¿Cómo es esto posible?
- ¿Qué efecto produce en la pila si mediante una instrucción se suma 4 al registro %esp?
- ¿Qué secuencia de **cuatro instrucciones de pila** se pueden ejecutar para **intercambiar los valores de dos registros**?