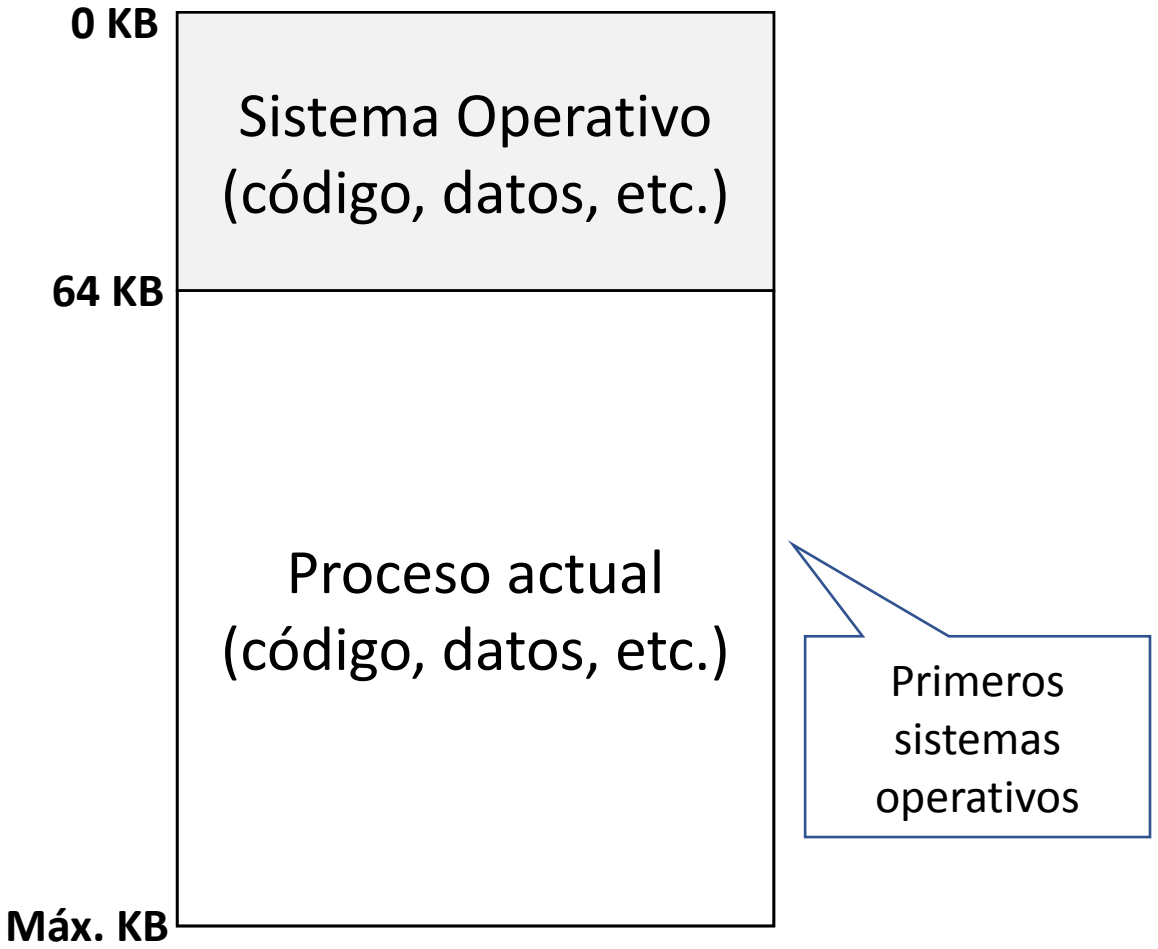
Administración de memoria

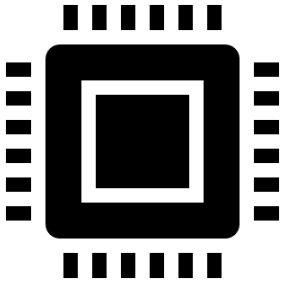
Adaptación (ver referencias al final)

Monotarea y multiprogramación



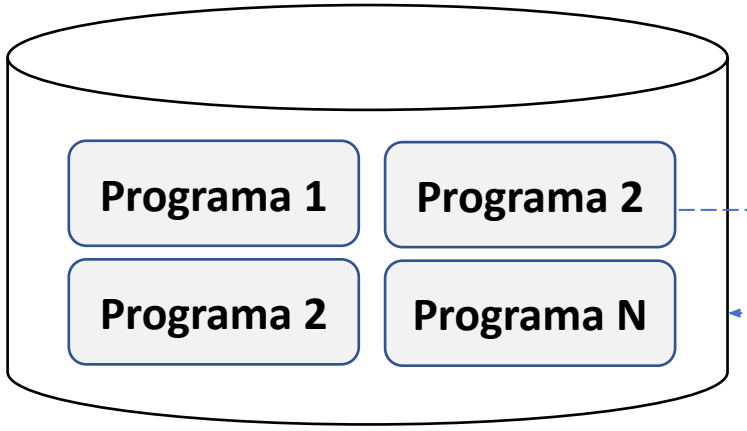
- Maximizar el uso del sistema
  - Varios procesos concurrentemente
- CPU compartida (en el tiempo) por varios procesos.
- Memoria principal (RAM) compartida (en el espacio) por varios procesos.
- ¿Qué implica compartir la memoria?
  - Gestión de memoria

1. Fetch
2. Decode
3. Execute



PC: 0x100

Apunta a la siguiente instrucción (en memoria) a ejecutar



Cargar en memoria para ejecutar

Expulsar de memoria

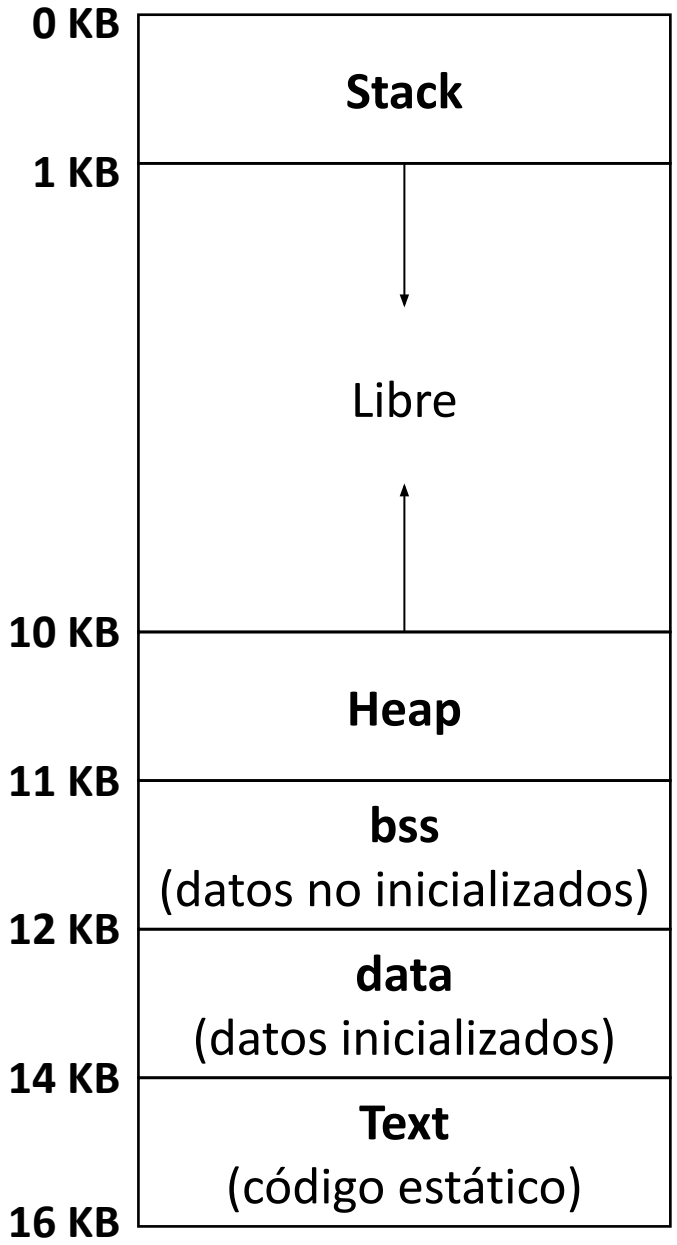
0 KB
Sistema Operativo
(código, datos, etc.)
64 KB
100 KB
MOV EAX, [0x70]
Proceso actual
(código, datos, etc.)
Máx. KB

¿Qué implicaciones tiene la expulsión?

Programas residen en disco (entidades estáticas)
Se ejecutan cuando se pueden cargar en memoria RAM

Espacio de direccionamiento

- Abstracción que crea el S.O
- Arreglo de bytes direccionables
- La abstracción da la idea de que el mapa de memoria inicia en la dirección 0 KB y termina en la dirección 16 KB
  - Físicamente esto no sucede así
- El proceso “cree” que ocupa toda la memoria disponible (16 KB) del sistema



loop.c

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    while(1) {
       printf("Infinito\n");
    }
    return 0;
}
```

hello.c

```
#include <stdio.h>

int main (int arcg, char* argv[])
{
    printf("Hello world!\n");
    return 0;
}
```

 # objdump -d -M i386 loop > loop.s

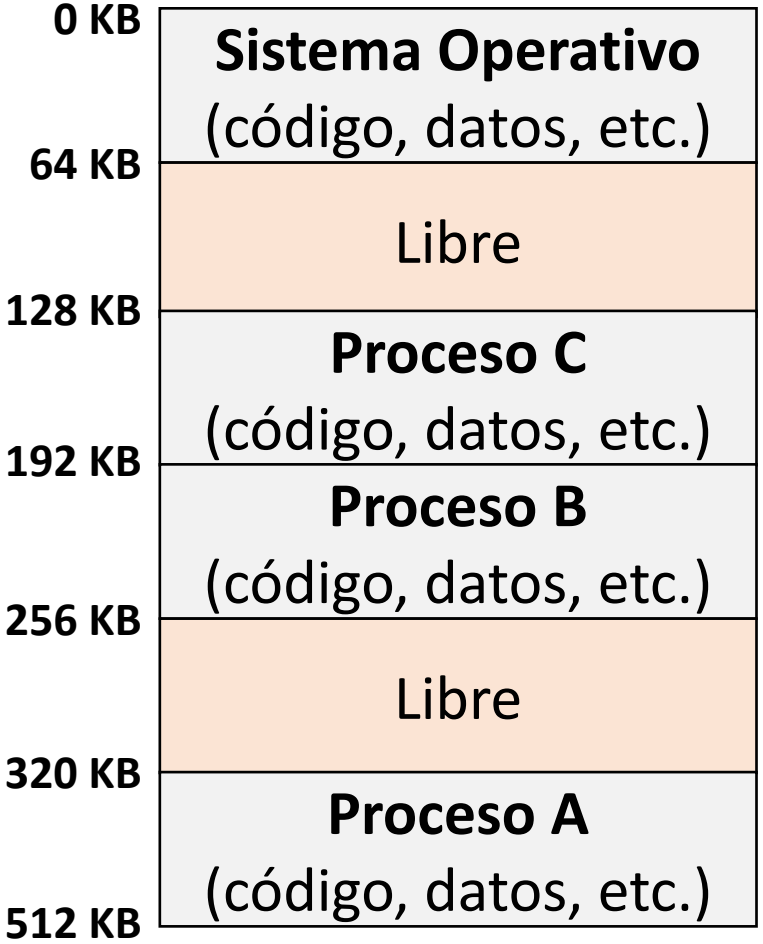
```
000000000040052d <main>:
  40052d: 55       push   %ebp
  40052e: 48       dec    %eax
  40052f: 89 e5    mov    %esp,%ebp
  400531: 48       dec    %eax
```

 # objdump -d -M i386 hello > hello.s

```
000000000040052d <main>:
  40052d: 55       push   %ebp
  40052e: 48       dec    %eax
  40052f: 89 e5    mov    %esp,%ebp
  400531: 48       dec    %eax
```

¿Misma dirección de memoria?
¿Qué sucede si los ejecuto a la vez? ¿Se solapan?

Multiprogramación



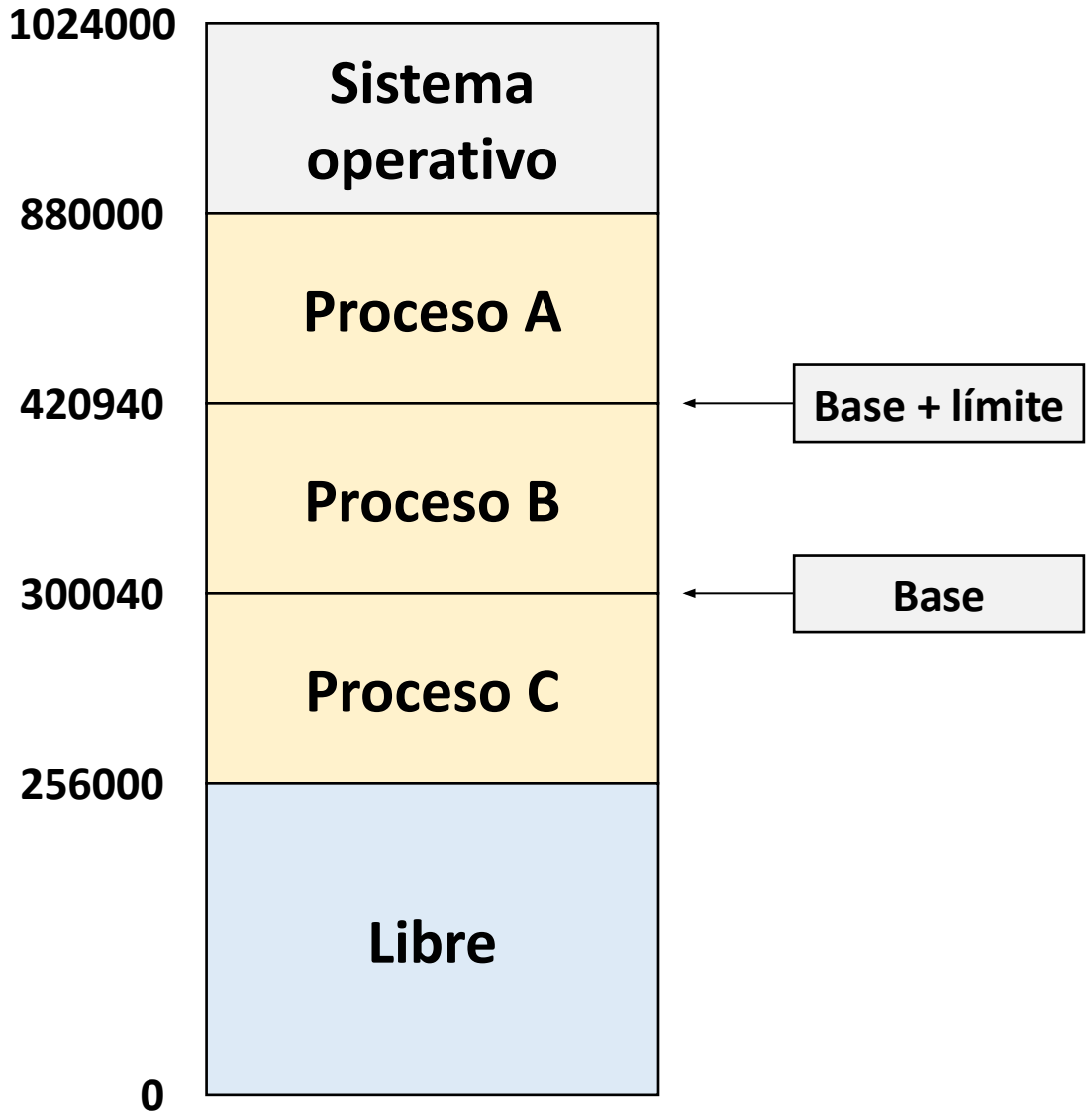
- Los procesos “creen” que tienen una máquina para ellos solos
  - CPU
  - RAM
- S.O debe permitir coexistencia **segura** de múltiples procesos en RAM
- **Físicamente** los procesos están en espacios de memoria diferentes

Objetivos de la gestión de memoria

- Ofrecer a cada proceso un espacio de direccionamiento lógico propio
  - Proceso se comporta como su propio espacio físico de direccionamiento
  - Separar los espacios de memoria de cada proceso
- Proporcionar protección entre los procesos
  - Operaciones de procesos NO pueden incidir en el espacio de direccionamiento de otros procesos
- Eficiencia
  - Tiempo: NO introducir retardos en la ejecución de los programas
  - Espacio: NO usar muchas estructuras de datos para la gestión de la memoria de cada proceso.
- **Para estos objetivos se requiere apoyo del hardware (MMU)**

Hardware

- El S.O no interviene en las operaciones de acceso de la CPU a memoria.
- Se necesita determinar el rango de direcciones válidas de cada proceso.
  - Operaciones de cada proceso deben hacer referencia al rango válido de su espacio de direccionamiento



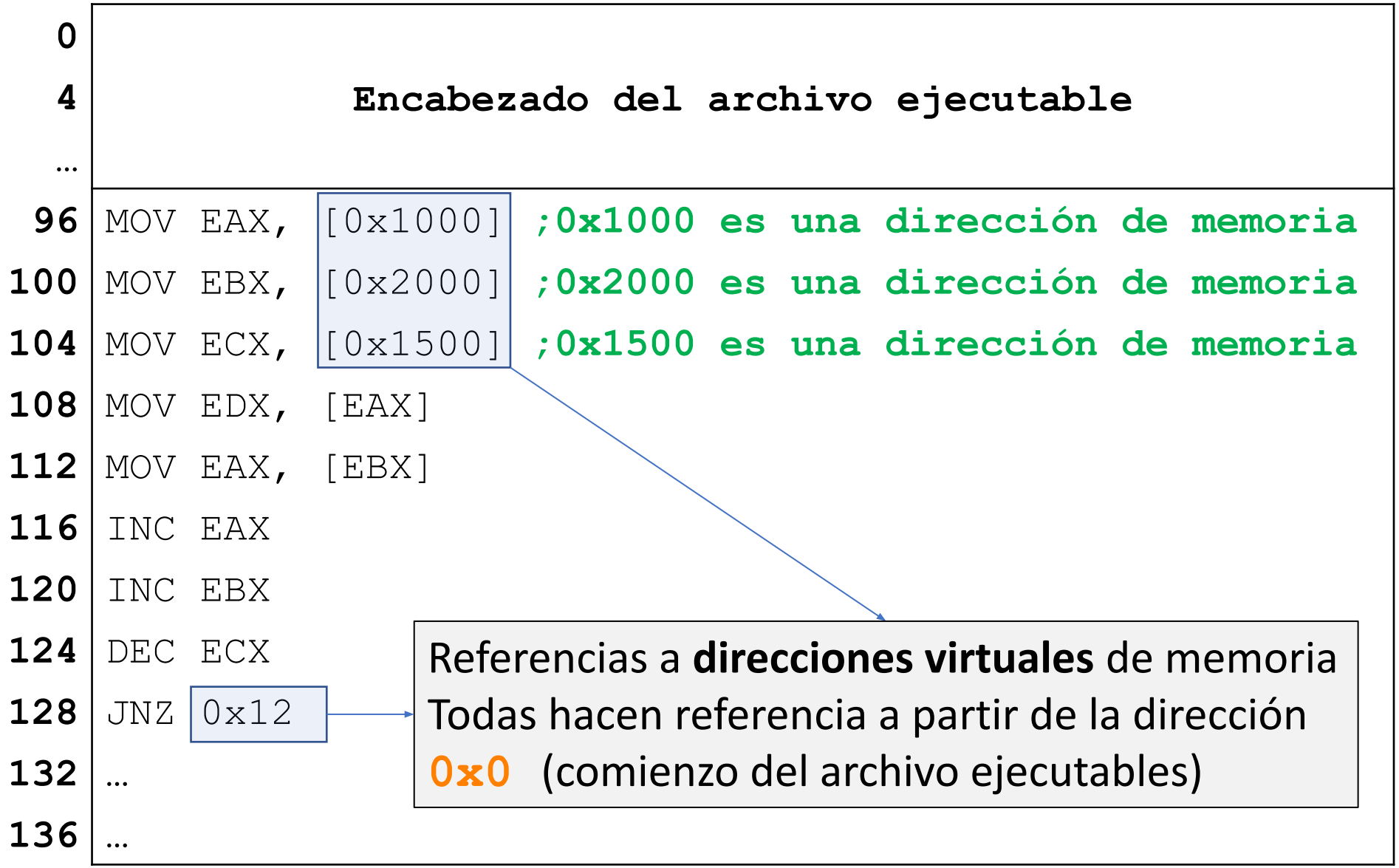
Hardware

- Hardware **MMU** (Memory Management Unit)
- Registros están implementados a nivel de hardware.
  - S.O es el único que tiene acceso a los registros.
- Registros controlan el espacio físico de direcciones.
- La protección la realiza la MMU comparando cada dirección generada en el espacio de usuario con los valores de los registros.

  - **Espacio de memoria virtual**
  - **Espacio de memoria física**

Traducción del direccionamiento

- En sistemas multiprogramados no se puede conocer ***a priori*** en donde será ubicado el proceso en memoria
  - Depende de la ocupación de memoria
- S.O.'s modernos permiten que los proceso de usuario residan en cualquier parte de la memoria física
  - La dirección `0x40052d` no necesariamente corresponde con la dirección física.
  - La dirección `0x40052d` es una **dirección virtual**.
- Direcciones en código fuente usualmente son simbólicas
  - `int i = 0;`

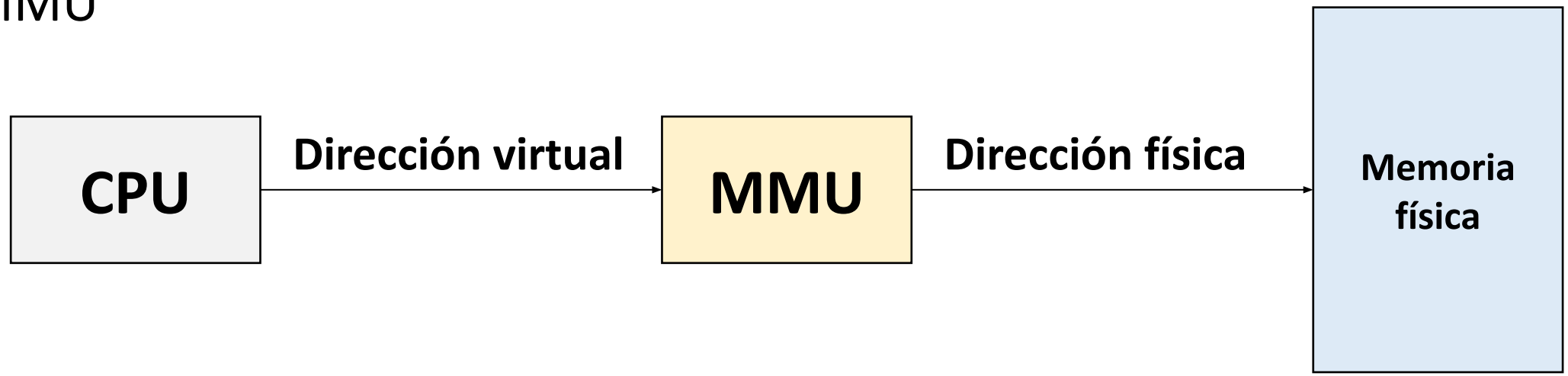

0
4
…
Encabezado del archivo ejecutable
96 MOV EAX, [0x1000] ;0x1000 es una dirección de memoria
100 MOV EBX, [0x2000] ;0x2000 es una dirección de memoria
104 MOV ECX, [0x1500] ;0x1500 es una dirección de memoria
108 MOV EDX, [EAX]
112 MOV EAX, [EBX]
116 INC EAX
120 INC EBX
124 DEC ECX
128 JNZ 0x12
132 …
136 …
Referencias a **direcciones virtuales** de memoria
Todas hacen referencia a partir de la dirección
0x0 (comienzo del archivo ejecutables)

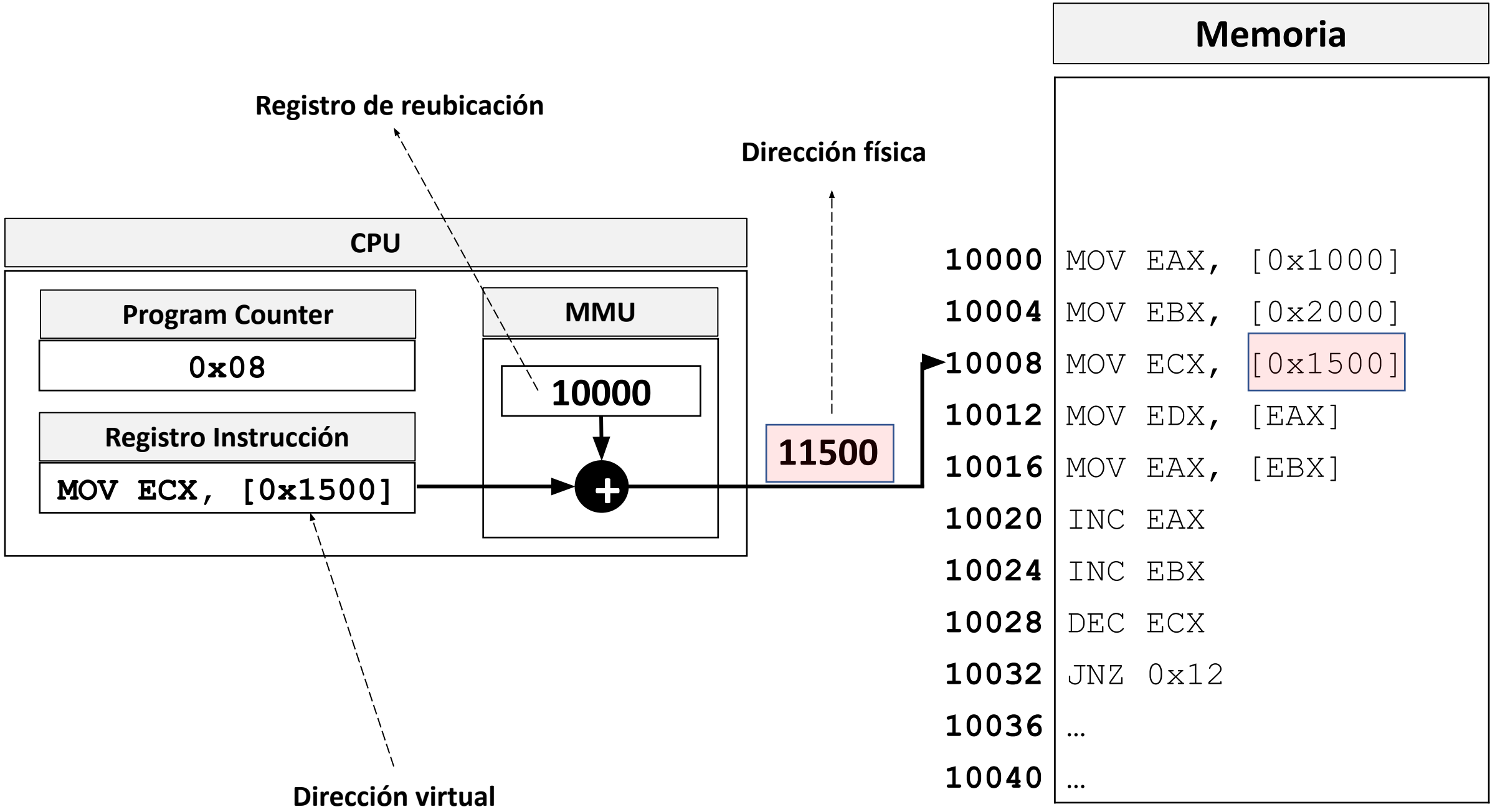
Traducción del direccionamiento

- Tiempo de compilación
  - Si se sabe a *a priori* donde va a estar el proceso, se genera código absoluto
  - Si la ubicación del proceso cambia, hay que recompilar
  - No hay traducción/mapeo: virtual corresponde con el físico.
- Tiempo de carga
  - Se debe generar código reubicable.
  - Si ubicación del proceso cambia se carga de nuevo el código para reflejar el cambio
- Tiempo de ejecución
  - La mayoría de S.O.'s modernos usan esta opción. Se necesita de la MMU
  - El mapeo/traducción no se hace hasta la ejecución del proceso.

Espacio de direccionamiento físico y virtual

- Direcciones lógicas/virtuales
  - Direcciones de memoria generadas por la CPU
- Direcciones físicas
  - Direcciones de memoria que ve la MMU
- La traducción virtual física se hace en tiempo de ejecución por la MMU




Memoria
Registro de reubicación
Dirección física
CPU
Program Counter
0x08
MMU
10000
Registro Instrucción
MOV ECX, [0x1500]
11500
10000 MOV EAX, [0x1000]
10004 MOV EBX, [0x2000]
10008 MOV ECX, [0x1500]
10012 MOV EDX, [EAX]
10016 MOV EAX, [EBX]
10020 INC EAX
10024 INC EBX
10028 DEC ECX
10032 JNZ 0x12
10036 …
10040 …
Dirección virtual

Demostración con gdb

- gdb <ejecutable>
- set disassembly-flavor intel
- layout next
- break main
- break *main+22
- break *0x40054d
- clear *0x40054d
- run
- info registers
- p/x $eax
- x/1dw 0x400535
- x/1dw $rbp
- info break
- ni
- si

Referencias

- Carretero Pérez, J., García Carballeíra, F., De Miguel Anasagasti, P., & Pérez Costoya, F. (2001). Gestión de memoria. In *Sistemas operativos. Una Visión Aplicada* (pp. 164–171). McGraw Hill.
- Silberschatz, A., Baer Galvin, P., & Gagne, G. (2018). Main Memory. In *Operating Systems Concepts* (10th ed., pp. 349–356). John Wiley & Sons, Inc.