

72.08 - Arquitectura de computadoras

| | |
|---|-----------|
| Análisis de binarios | 4 |
| Tipos de arquitecturas:..... | 4 |
| Compilación y link-edicion en C:..... | 4 |
| Ejecución de un programa:..... | 4 |
| Programa en memoria..... | 4 |
| Programa en disco..... | 5 |
| Programa visto desde Bless..... | 5 |
| Llamar al Sistema operativo..... | 6 |
| Syscalls..... | 6 |
| Registros de Intel..... | 6 |
| Arquitectura 8088/8086 (16 bits)..... | 7 |
| Arquitectura de 80386..... | 7 |
| Programa de memoria..... | 8 |
| Assembler de intel | 9 |
| registros en 80386(32 bits)..... | 9 |
| Sintaxis..... | 9 |
| Stack..... | 11 |
| Int 80h..... | 11 |
| .bss..... | 11 |
| .text..... | 12 |
| Assembler 64 bits..... | 12 |
| Tipo de datos..... | 13 |
| Asm y C | 15 |
| Repaso de la Stack (pila)..... | 15 |
| Instrucciones RET y CALL..... | 17 |
| Pasaje de parámetros a C..... | 17 |
| dump de GDB 32bits..... | 17 |
| Stack de programa compilado en gcc..... | 18 |
| Memoria y Entrada y Salida | 19 |
| Sistema de Entrada y Salida..... | 19 |
| CPU..... | 19 |
| Bus de datos..... | 19 |
| Bus de direcciones..... | 19 |
| Registros de intel..... | 20 |
| Bytes..... | 20 |
| Mapa de memoria..... | 20 |
| Memorias..... | 21 |
| Integrados Compuertas y Decodificadores..... | 22 |
| Registros..... | 22 |
| Compuertas..... | 22 |

| | |
|--|-----------|
| Decodificación de Hardware..... | 22 |
| Chip select..... | 23 |
| IO/M patita..... | 23 |
| R/^W patita..... | 23 |
| Periféricos..... | 24 |
| Pines básicos de control..... | 25 |
| Decodificación de hardware..... | 25 |
| Sistemas de entrada y salida..... | 25 |
| Mapeo de memoria..... | 27 |
| Ejecución de una instrucción:..... | 27 |
| Interrupciones | 28 |
| Sistema de Entrada y Salida..... | 28 |
| Rutina de atención de la interrupción (Drivers de interrupción)..... | 28 |
| Tipos de interrupciones..... | 29 |
| PIC (Controlador programable de interrupciones)..... | 30 |
| BIOS..... | 31 |
| IRQ (Interrupt Request Queue)..... | 32 |
| Excepciones..... | 32 |
| IDT Loader..... | 33 |
| Modo Protegido..... | 34 |
| ¿Qué es?..... | 34 |
| Comunicación de tareas..... | 34 |
| Protección de tareas → MMU memory management unit..... | 36 |
| Descriptor Table..... | 36 |
| Descriptores..... | 37 |
| Dirección lineal → Memoria Virtual..... | 39 |
| Carga de IDT..... | 39 |
| Interrupciones en modo protegido..... | 41 |
| Tabla de excepciones..... | 41 |
| Privilegios de Entrada y Salida..... | 41 |
| Memoria Virtual..... | 42 |
| Direccionamiento virtual..... | 42 |
| Memoria virtual para múltiples procesos..... | 42 |
| Elección de Intel para 32 bits..... | 43 |
| Paginación..... | 44 |
| Memoria Caché..... | 46 |
| Memoria Caché..... | 46 |
| Etiquetas y bloques..... | 47 |
| ¿Cómo funciona?..... | 47 |
| Procesadores de 64 bits | 48 |
| ARM..... | 49 |
| Arquitectura y familias..... | 49 |
| Modos:..... | 49 |
| Set de registros:..... | 49 |

| | |
|---|-----------|
| Pipeline en ARM7..... | 50 |
| Arquitecturas Vonn Neumann y Harvard..... | 51 |
| TDMI..... | 52 |
| Thumb..... | 52 |
| Jazelle..... | 52 |
| ARM9TDMI..... | 53 |
| GPU (Unidades de procesamiento gráfico)..... | 53 |
| Computación Paralela..... | 53 |
| Threads..... | 53 |
| GPU..... | 54 |
| CUDA..... | 54 |
| Paralelización..... | 54 |
| Pasos para ejecutar código en GPU..... | 54 |
| Grids y Blocks..... | 55 |
| Machine Learning..... | 55 |

Análisis de binarios

Tipos de arquitecturas:

| PC | Mac | Smartphone | |
|-------------------------|------------------------------|------------|-----------------|
| variedad de fabricantes | un fabricante | iphone | android → linux |
| procesador amd64 | procesador mac silicon arm64 | | |

Compilación y link-edicion en C:

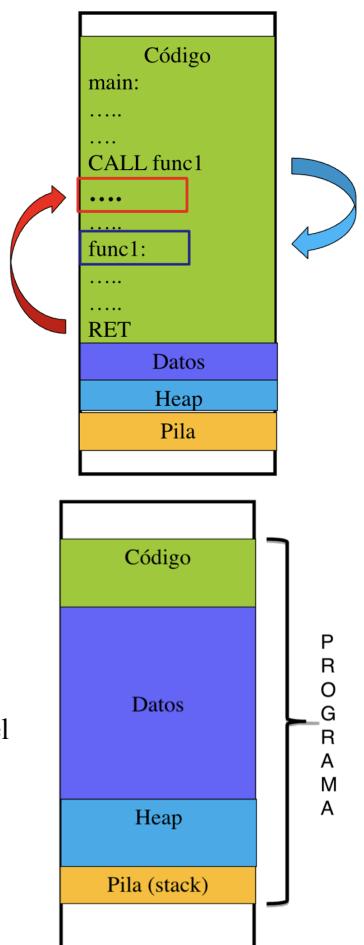
1. gcc primero invoca al pre-procesador cpp, el cual toma el código fuente original y realiza la macro-expansión de directivas como #include y #define
2. gcc toma el control de es salida y convierte el código en C en código equivalente en Assembler → genera un archivo con extensión .s
 - a. en código asm puede ser generado en sintaxis AT&T o en la sintaxis Intel
3. el compilador GNU de assembler “gas” se encargar de generar el código objeto → extensión .o
4. en linux el ejecutable podrá ser de diferentes formatos → se debe al link editor ld, lo cual, genera el binario
 - a. elf
 - b. a.out

Ejecución de un programa:

1. Línea de comando (bash)
2. Sistema operativo (execve)
3. Lectura de disco
4. Asignacion de espacio en memoria → asi facilita la llamada recurente de las funciones, entonces al copiarlo en la memoria permite un acceso mas rapido
5. Call o Jmp a direccion de memoria de programa
 - a. Call → instrucion de asm que permite llamar a una función y guarda el valor de retorno en la pila (RET)
 - b. jmp → salta a la posicion de memoria indicado en la instrucción
6. Ejecucion de programa

Programa en memoria

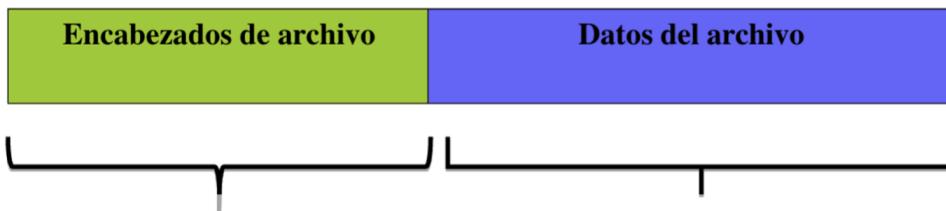
- código → instrucciones del programa
- datos → variables estáticas y globales que se inician al cargar el programa



- heap → memoria dinámica que se reserva y se libera en tiempo de ejecución
 - tiempo de carga
 - RAM
 - en C sería el ‘malloc’ y ‘free’
- stack → argumentos y variables locales a la función
- porción de memoria que se guardan los variables

cada segmento menos heap contiene un puntero

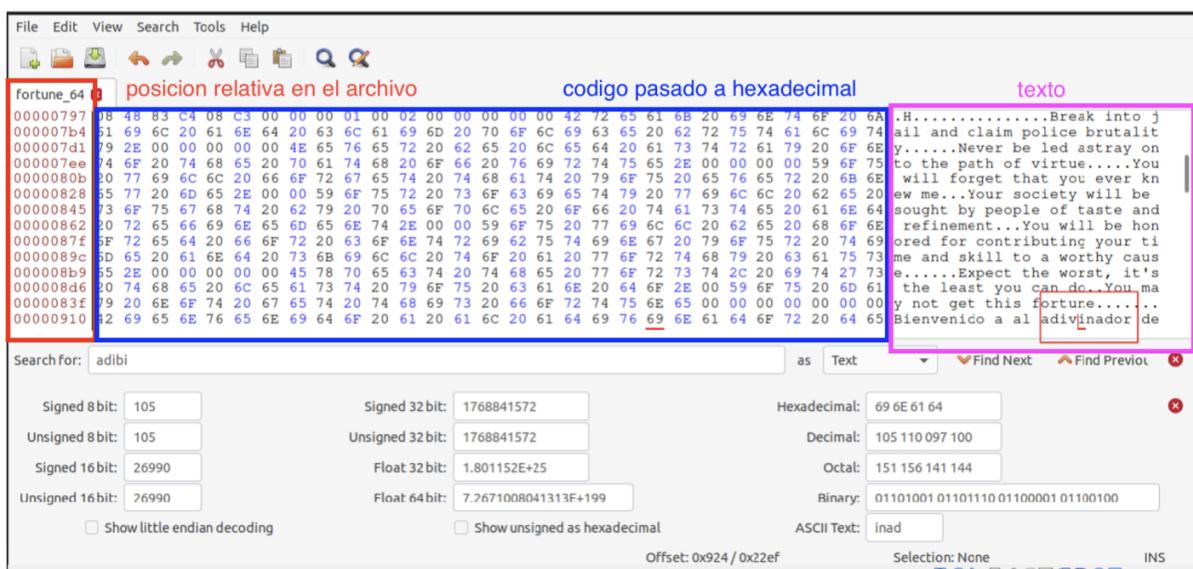
Programa en disco



- Encabezados de programa (Segmentos)
- Encabezado de Secciones
 - (Secciones: text, data, rodata, etc)
- Código
- Datos

encabezado del archivo → es para el sistema operativo

Programa visto desde Bless



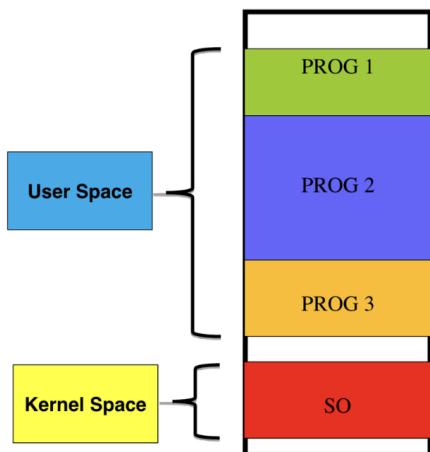
como se ve que el error de ortografía esta hecho en texto podemos modificar el bit

```

osboxes@osboxes:~/arqui/ArquiTP1$ ./fortune_64
Bienvenido a al adivinador de la fortuna! Este mensaje es muy lar
go, y puede ser muy molesto al usuario. Tal vez deberíamos acorta
rlo?
Cual es tu nombre?: 

```

Llamar al Sistema operativo



PRIMER PROGRAMA que se corre es la bios → determina que se va correr cuando que prende la maquina

userspace → la parte que interactúa con el usuario

kernel space → la parte donde funciona el sistema operativo así el usuario no puede modificarlo y romper el funcionamiento de la máquina

Syscalls

<https://syscalls32.paolostivanin.com/>

- Son interfaces proporcionadas por el núcleo del sistema operativo para permitir que las aplicaciones realicen solicitudes de servicios al sistema operativo.

```

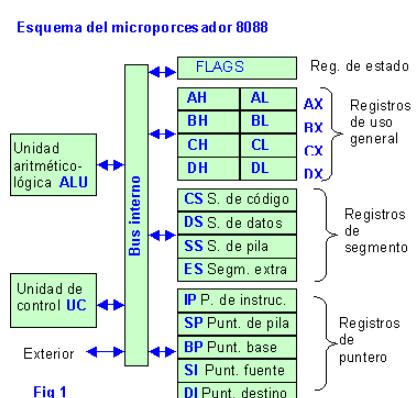
push ebp ;armo stack
mov ebp,esp ;creo un stack frame asi no rompo stack

push #; backup de registro asi no rompo en stack
...
mov eax, # ; # es el numero de syscall
mov ebx, #
... ; preparo los registros con los argumentos indicados para el syscall
int 80h ; interrupcion 80 le indica al SO que busque en los syscalls
        ;y corra la funcion

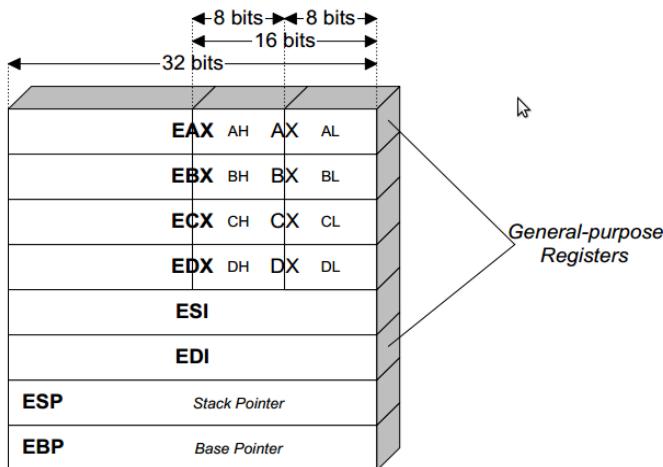
...
mov esp,ebp ;desarmo stackframe
pop ebp
  
```

Registros de Intel

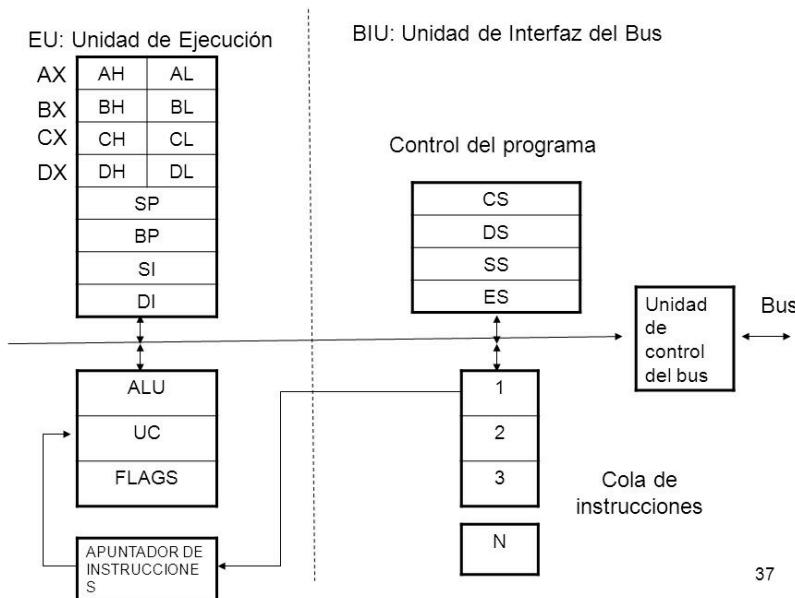
- IP: Instruction Pointer (EIP en 32 bits y RIP en 64 bits):
Puntero a las próxima instrucción a ejecutarse.



- SP: Stack Pointer (ESP en 32 bits y RSP en 64 bits): Puntero a la pila para guardar y extraer datos.
- Registros de Manejo de Memoria:
 - CS: Code Segment.
 - DS: Data Segment. (también ES, FS y GS)
 - SS: Stack Segment.
- Los registros de segmentos mantienen su tamaño en arquitectura de 32 y 64 bit



Arquitectura 8088/8086 (16 bits)

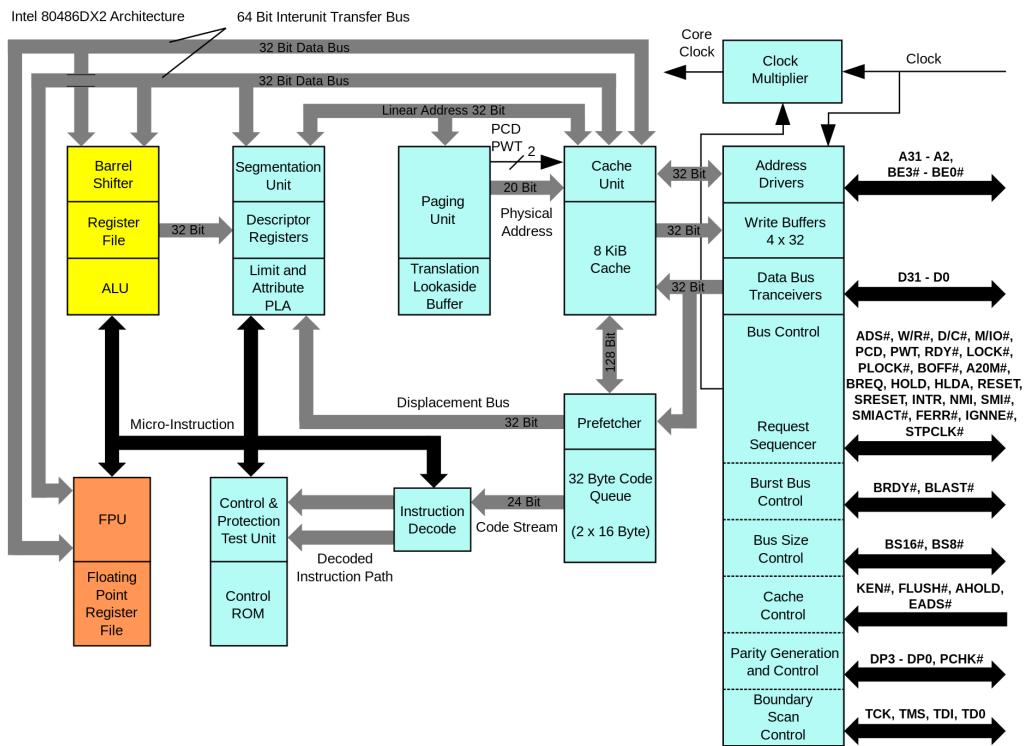


37

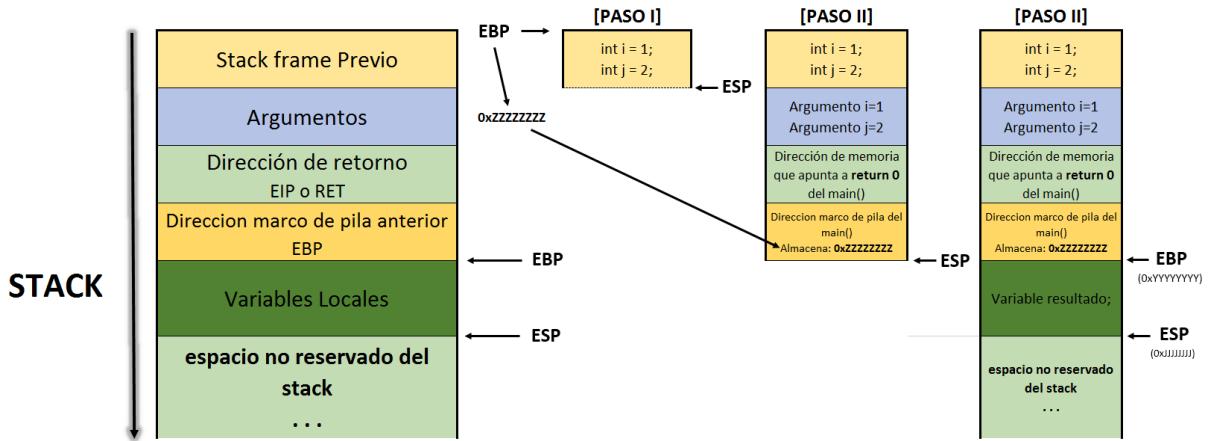
Arquitectura de 80386

- **multitarea:** existe un requisito importante para los sistemas operativos multitarea que es tener espacio de memoria individual para cada tarea, y un espacio de memoria común para varias tareas
- **multusuario:** que más de un usuario tenga acceso a la CPU, lo que genera más tareas
- **tiempo compartido:** el SO asigna un tiempo para cada tarea (time slot)
- **tiempo real:** la conmutación de tareas viene dada por acontecimiento externos
- **sistema de protección:** mínimo dos niveles, de usuarios y de supervisor.

- memoria virtual
 - las memorias RAM o ROM siguen siendo caras si se las compara con los precios de los discos rigidos



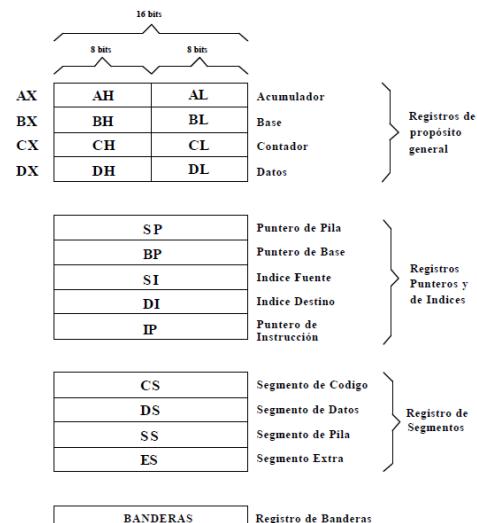
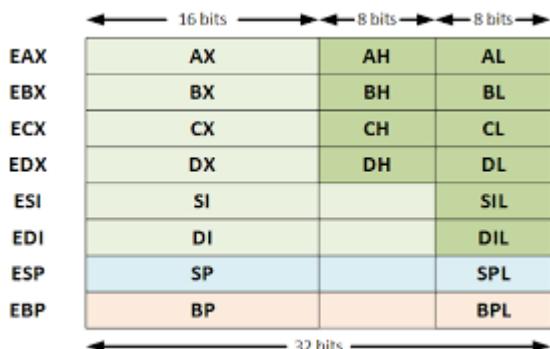
Programa de memoria



Assembler de intel

<https://www.nasm.us/doc/nasmdoc3.html>

registros en 80386(32 bits)



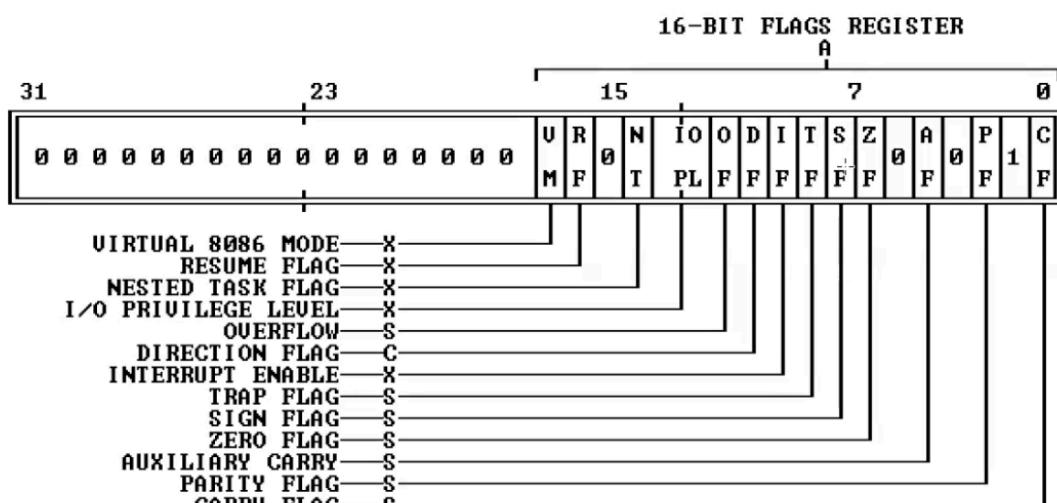
Sintaxis

modos de direccionamiento

| Modos de direccionamiento | |
|-----------------------------|---|
| mov destination, source | mov → copy osea hace destination=source |
| mov registro, [dirección] | direccionamiento directo/absoluto |
| mov registro, [registro] | direccionamiento indirecto |
| mov registro,[registro + #] | direccionamiento con índice |
| mov [registro], [registro] | NO HACER NUNCA |

flags

Figure 2-8. EFLAGS Register



S = STATUS FLAG C = CONTROL FLAG X = SYSTEM FLAG

NOTE: 0 OR 1 INDICATES INTEL RESERVED. DO NOT DEFINE.

ejemplos

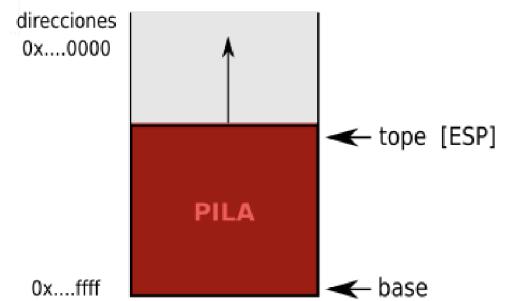
| | | | | | |
|------------------------------------|---|---|---------------|-----------------------|--------------|
| <i>section .text</i> | <i>donde estan las instrucciones</i> | | | | |
| <i>global _start</i> | <i>funciones globales</i> | | | | |
| <i>_start:</i> | <i>int main(char **args) de c</i> | | | | |
| <i>mov dx, 0FFh</i> | <i>inicializa registros</i> | | | | |
| <i>mov bx, 20h</i> | | | | | |
| <i>add dx, bx</i> | <i>suma en dx = dx + bx</i> | | | | |
| <i>push dx</i> | <i>suma a la pila dos valores</i> | | | | |
| <i>push 4</i> | <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">4</td><td style="padding: 5px;"><i>arriba</i></td></tr> <tr> <td style="padding: 5px;"><i>dx osea bx+ dx</i></td><td style="padding: 5px;"><i>abajo</i></td></tr> </table> | 4 | <i>arriba</i> | <i>dx osea bx+ dx</i> | <i>abajo</i> |
| 4 | <i>arriba</i> | | | | |
| <i>dx osea bx+ dx</i> | <i>abajo</i> | | | | |
| <i>pop cx</i> | <i>agarra lo ultimo de la pila y se lo asigna a cx [cx] = 4</i> | | | | |
| <i>ciclo:</i> | <i>loop</i> | | | | |
| <i>inc bx</i> | <i>incrementa bx</i> | | | | |
| <i>dec cx</i> | <i>decrementa cx</i> | | | | |
| <i>jnz ciclo</i> | <i>jump not zero → sigue hasta que cx sea 0</i> | | | | |
| <i>mov eax, parámetros</i> | | | | | |
| <i>mov ah, [parametro]</i> | | | | | |
| <i>mov bl, [parametro]</i> | | | | | |
| <i>add ah,bl</i> | | | | | |
| <i>mov [salida], ah</i> | | | | | |
| <i>ret 0</i> | <i>“return 0” de c</i> | | | | |
| <i>section .data</i> | <i>dónde están las variables de datos, variables globales, y variables estaticas</i> | | | | |
| <i>parámetros db 11h, 12h, 13h</i> | <i>db tamaño de dato</i> | | | | |
| <i>salida db 0</i> | | | | | |

Stack

- estructura de datos preservada en una región distinta de la memoria que permite, entre otras cosas, que en nuestros métodos llame a otros métodos (funciones) y continúe sus ejecuciones tan pronto como se produce un retorno, preservando las variables locales

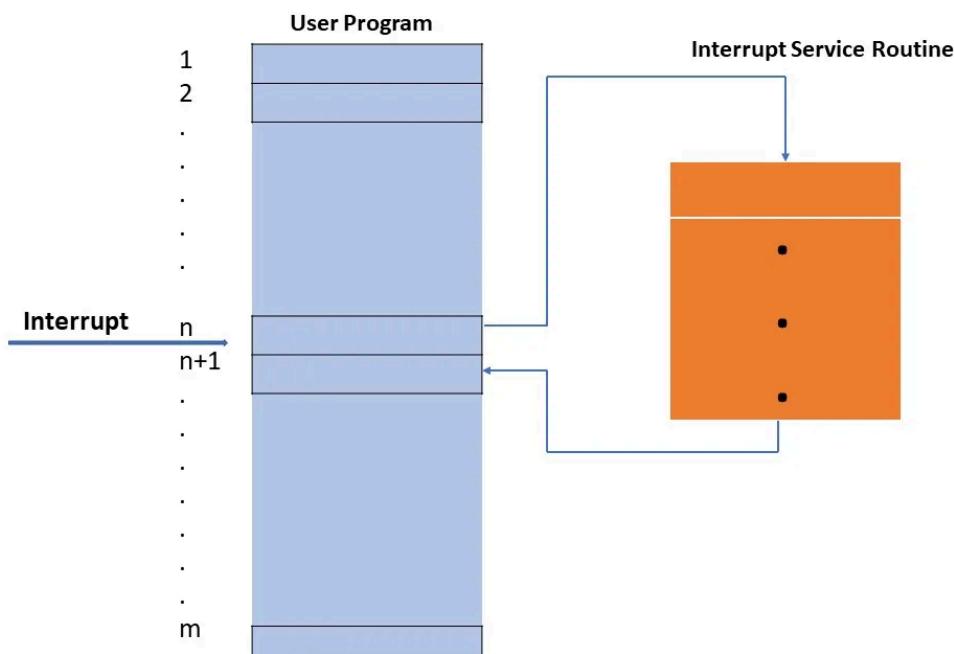
```
push ebp  
mov ebp, esp  
...  
mov esp, ebp  
pop ebp
```

stackframe → prolijidad
ayuda a no romper el stack del SO



Int 80h

- Las instrucciones de entrada estándar, sirven para leer caracteres desde el teclado, y las instrucciones de salida estándar muestran caracteres en la pantalla
- Sistema operativo hace una salida default



.bss

- declarando datos inicializados

```
label:      instruction operands ; comment
```

- definición de constantes

```
string          db      'hello, world'  
length         equ     $-string
```

.text

- creando loops

```
label1 ; some code

.loop
    ; some more code
    jne     .loop
    ret

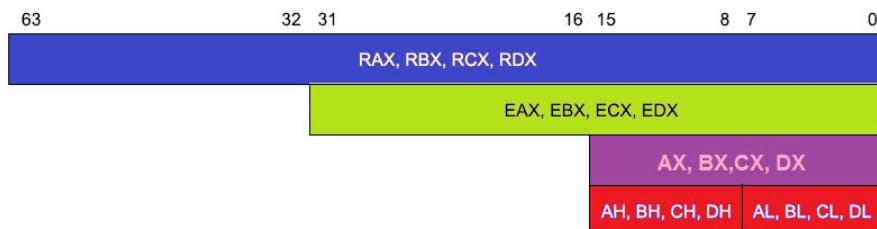
label2 ; some code

.loop
    ; some more code
    jne     .loop
    ret
```

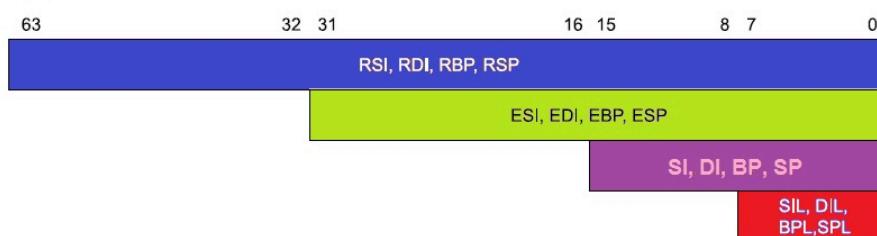
Assembler 64 bits

La diferencia principal entre el pasaje de parámetros en una arquitectura de 32 bits y una de 64 bits radica en la cantidad de registros disponibles y en cómo se utilizan estos registros para pasar parámetros

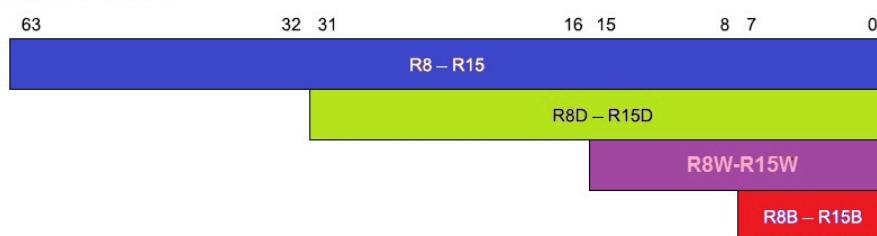
- pasaje de parámetros en orden
 - **RDI, RSI, RDX, RCX, R8, R9 luego se pasa por la pila**



Registros RSI, RDI, RBP, RSP

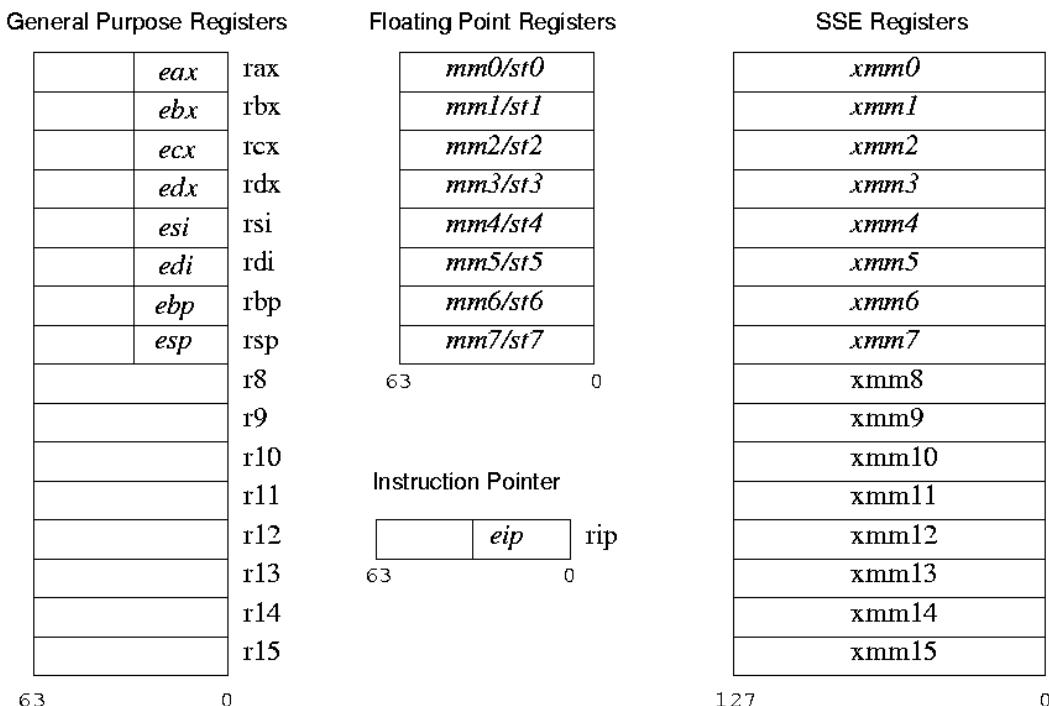


Registros R8-R15



Pasaje de argumentos:

- Si el dato es **INTEGER**, se van ocupando los registros **rdi, rsi, rdx, rcx, r8 y r9** en orden
- Si el dato es **SSE**, se van ocupando los registros **xmm0 a xmm7** en orden.
- Si el dato es **MEMORY**, se **pasan por stack** y devuelven por **stack**, de izquierda a derecha (igual que en 32 bits).



Valores de retorno:

- Si el dato es **INTEGER**, se utiliza **rax** y **rdx**
- Si el dato es **SSE**, se retorna por **xmm0** y **xmm1**

Registros que se preservan: → hay que hacer back-up si los uso

- **rbp**
- **rsp**
- **rbx**
- **r12**
- **r13**
- **r15**

Tipo de datos

en section .data

| | | |
|-----------|---------|--|
| db | 1 byte | chars |
| dw | 2 bytes | números enteros cortos, valores de código de caracteres Unicode básico. |
| dd | 4 bytes | enteros, direcciones de memoria de 32 bits |
| dq | 8 bytes | enteros largos, direcciones de memoria de 64 bits, valores de punto flotante de doble precisión. |

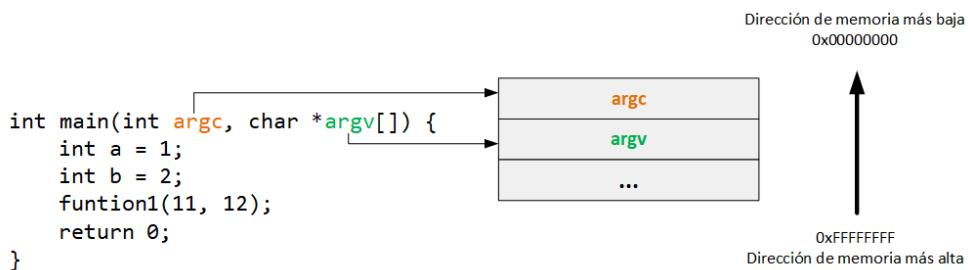
en section .bss

| | | |
|-------------|---------|--|
| resb | 1 byte | depende del número que sigue cuantos lugares reservaste resb cantidad_de_bytes. |
| resw | 2 bytes | |
| resd | 4 bytes | |
| resq | 8 bytes | |

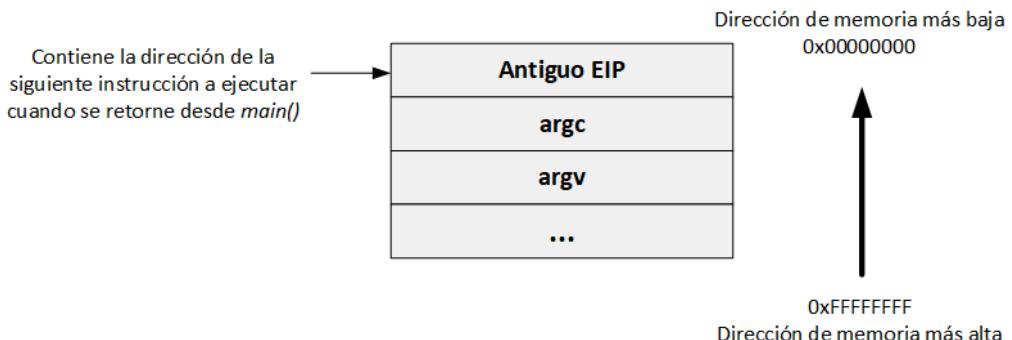
Asm y C

Repaso de la Stack (pila)

una función vista desde la pila

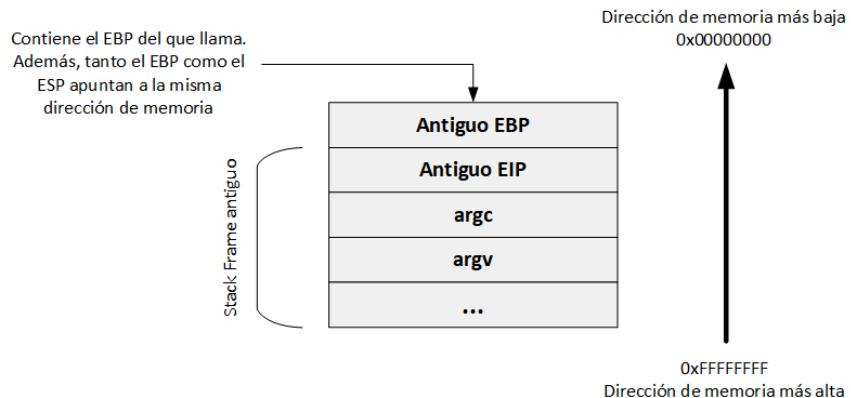


1. al iniciar este programa, primero hay un pasaje de los parámetros del main (argc y argv), son agregados al **stack (de derecha a izquierda)**
2. Se llama la función main y la CPU realiza un PUSH del contenido del EIP en el stack, y señala el primer byte después de la instrucción CALL
 - a. se necesita saber la dirección de la próxima instrucción para poder proceder cuando **regresemos de la función llamada**

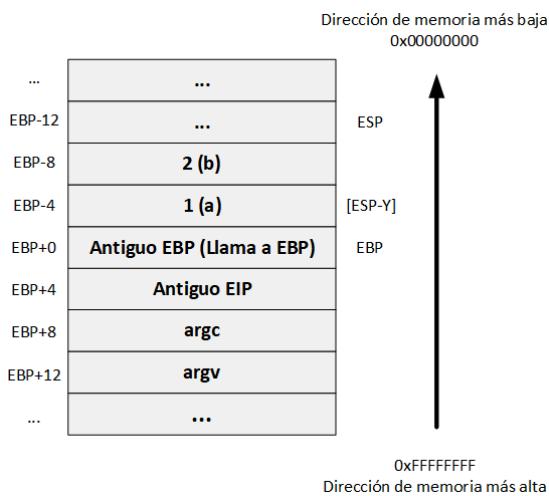


3. El que llama (en este caso el sistema operativo) “ pierde el control”, y el que es llamado (función main) toma el control
4. Como estamos en la función main, se crea un nuevo stack frame, el cual, es definido por el EBP y el ESP; almacenando el EBP actual en el stack, con el fin de poder saber que estamos

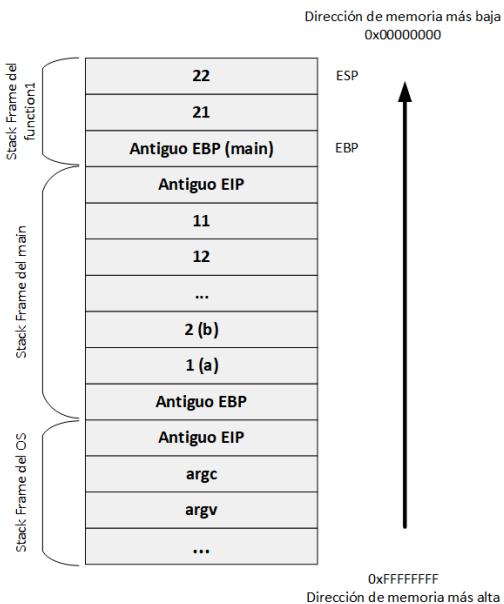
viviendo a la función que llamó a main. Cuando el valor del EBP es almacenado, este es actualizado, y ahora apunta a la parte superior del stack (PILA CRECE HACIA ABAJO)



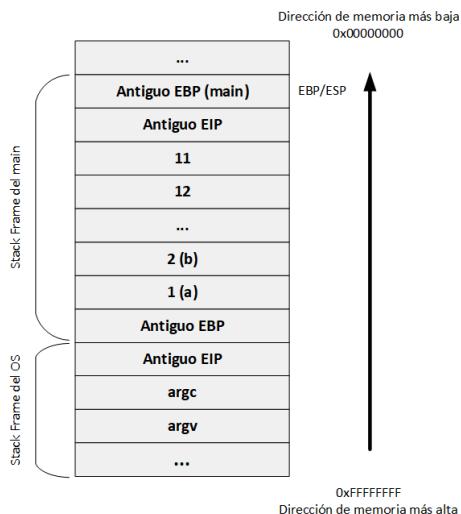
5. Se completa el stack frame de main, y las variables locales son copiadas en el stack.



6. cuando se llama la funcion1, se pasan los parámetros de izquierda a derecha



7. cuando se termina de ejecutar la funcion1 se vuelve al stack antiguo y “se liberan los lugares” de memoria antes ocupados



Instrucciones RET y CALL

- Cuando se ejecuta una instrucción RET, el procesador toma el contenido de los apuntado por ESP y salta a esa posición de memoria
- La instrucción **CALL** guarda en la pila la próxima instrucción a ejecutarse ó también llamada **dirección de retorno**. La función llamada (func1) tiene que dejar la pila sin modificar antes de terminar, así RET puede volver correctamente.

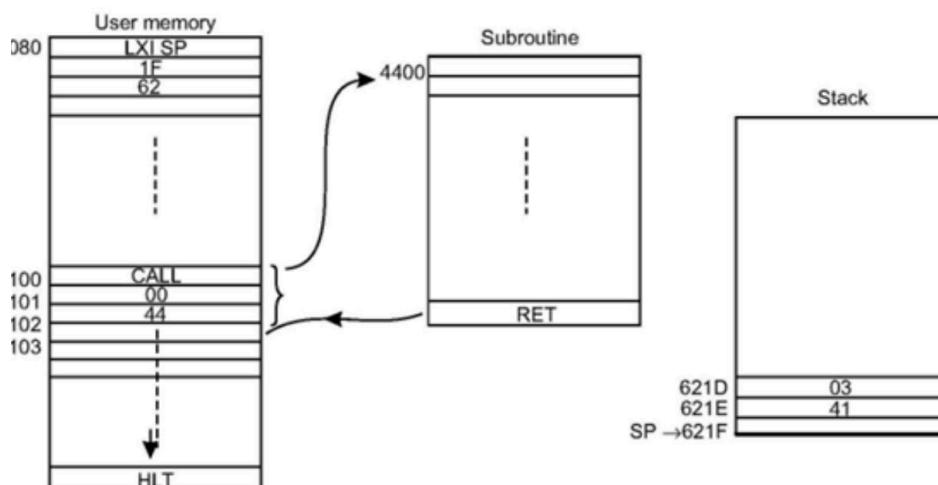


Fig. 6.2: Use of CALL-RET in a subroutine

Pasaje de parámetros a C

- Arquitectura de 32 bits → Se pasan por la pila
- Arquitectura de 64 bits → Se pasan primero por registros y luego por la pila
 - Para pasar argumentos se usan los registros RDI, RSI, RDX, R10, R8, R9.
 - Si la función necesita más parámetros se usa la pila
 - Para punto flotante (float, double), xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7

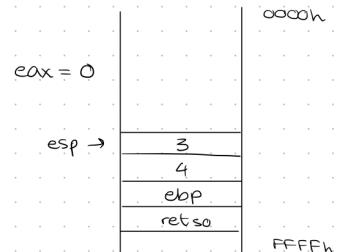
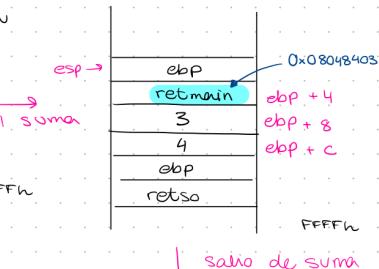
dump de GDB 32bits

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x080483e9 <+0>: push ebp
0x080483ea <+1>: mov ebp,esp
0x080483ec <+3>: sub esp,0x8
0x080483ef <+6>: mov DWORD PTR [esp+0x4],0x4
0x080483f7 <+14>: mov DWORD PTR [esp],0x3
0x080483fe <+21>: call 0x80483dc <suma>
0x08048403 <+26>: mov eax,0x0
0x08048408 <+31>: leave
0x08048409 <+32>: ret
End of assembler dump.
(gdb) disassemble suma
Dump of assembler code for function suma:
0x080483dc <+0>: push ebp
0x080483dd <+1>: mov ebp,esp
0x080483df <+3>: mov eax,DWORD PTR [ebp+0xc] 4
0x080483e2 <+6>: mov edx,DWORD PTR [ebp+0x8] 3
0x080483e5 <+9>: add eax,edx eax = 7
0x080483e7 <+11>: pop ebp
0x080483e8 <+12>: ret
End of assembler dump.
(gdb)
```

si se cambia el main pero la función suma se mantiene igual

Dump of assembler code for function main:

```
0x080483e9 <+0>: push ebp
0x080483ea <+1>: mov ebp,esp
0x080483ec <+3>: sub esp,0x18
0x080483ef <+6>: mov DWORD PTR [esp+0x4],0x4
0x080483f7 <+14>: mov DWORD PTR [esp],0x3
0x080483fe <+21>: call 0x80483dc <suma>
0x08048403 <+26>: mov DWORD PTR [ebp-0x4],eax
0x08048406 <+29>: mov eax,0x0
0x0804840b <+34>: leave
0x0804840c <+35>: ret
End of assembler dump.
```



cuando trabajo con strings con dimension sin instanciar y luego relleno sin chequeo de dato valido

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int pass = 0;
    char buff[15];

    printf("\nEnter the password : \n");
    gets(buff);

    if(strcmp(buff, "thegeekstuff")!=0)
    {
        printf ("\n Wrong Password \n");
    }
}
```

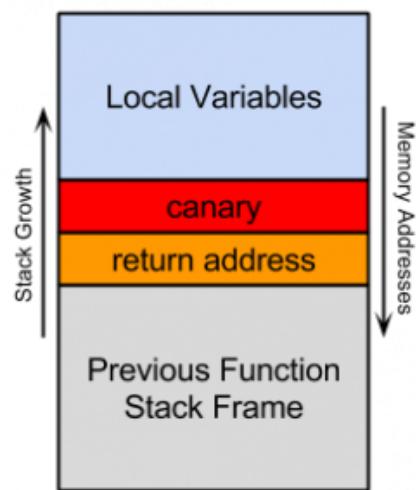
se reserva 15 en buff
 ↓
 tema del gets no restringue
 al password pues gets
 espera un string que
 termina en cero.
 ↓
 si me paso del lugar reservado
 voy a pisar el valor de
 retorno.
 ↓
 osea va saltar a es lugar
 de memoria

| | |
|-------|--------------------|
| buff | 111111111111111111 |
| pass | 111111111111111111 |
| ebp | 111111111111111111 |
| retso | 111111111111111111 |

gcc → crea un canary valor random que hace un chequeo antes de retornar para ver si se rompio el stack o no

Stack de programa compilado en gcc

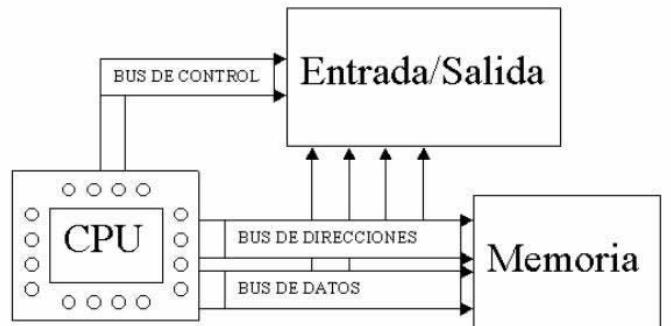
- contiene un canary
 - También conocida como **Stack Cookie**. Es una implementación de seguridad que coloca un valor **al lado de la dirección de retorno en el stack**.
 - **stack protector** → si está activado entonces el stack del programa compilado en gcc contiene un canary después de ret de la función y después del ebp
- **stack_chk_fail**: la función de gcc
 - Ubica un valor entre EBP y las variables locales que se lo denomina CANARY
 - Antes de retornar verifica que el CANARY no haya sido modificado
 - Si lo fue termina la ejecución.



Memoria y Entrada y Salida

Sistema de Entrada y Salida

- Están todos los periféricos. El procesador manda la instrucción sin saber si es memoria o IO. Luego la patita IO/M se encarga de eso. Solo pueden mandar información al procesador de a uno a la vez
- **Bus de dirección**: Manda la dirección de memoria con la que el cpu quiere intercambiar info.
- **Bus de datos**: Manda la información a dicha zona de memoria. La info puede ir para los dos lados.



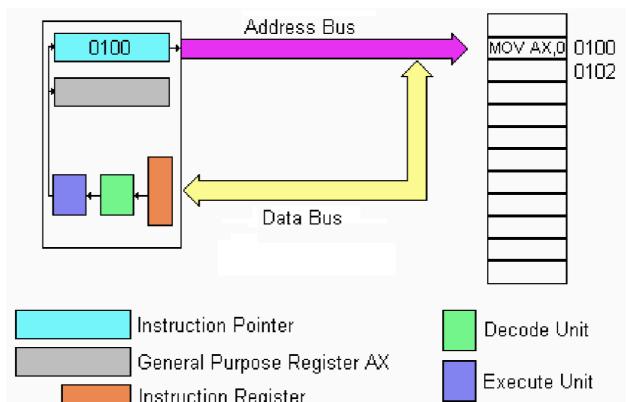
CPU

- **unidad de control**: recupera instrucciones de memoria, las decodifica, escribe en memoria
- **unidad de ejecución**: lleva a cabo la ejecución de la instrucción
- **registros**: memoria interna utilizada como variable
- **flags**: indican eventos luego de ejecutar las instrucciones

Bus de datos

- bi-direccional

- cada cable del bus de datos contiene un bit de la instrucción a ejecutar
- hoy en dia lo máxima cantidad de bus de datos es 64 bits
- Se encarga de **transmitir los datos** entre los **diferentes componentes** de la computadora



- la CPU, la memoria RAM, los dispositivos de entrada y salida, etc.
- **El bus de datos es el medio por el cual la información fluye dentro del sistema.**
 - Por ejemplo, cuando copias un archivo de tu disco duro a la memoria RAM, los datos se mueven a través del bus de datos.

Bus de direcciones

- es **unidireccional**
- Transfiere los datos, el bus de direcciones se encarga de señalar dónde se encuentran esos datos en la memoria.
- Cuando la CPU necesita acceder a ciertos datos en la memoria, envía una dirección a través del bus de direcciones para **indicar la ubicación específica** de esos datos en la memoria. Por lo tanto, el bus de direcciones es crucial para el proceso de lectura y escritura en la memoria.

Registros de intel

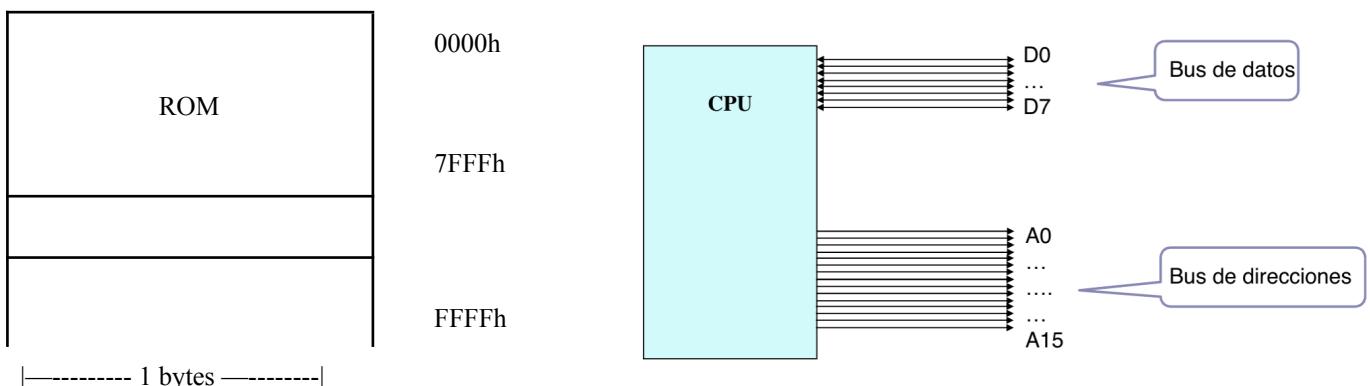
- IP: puntero a instrucción
 - primer instrucción que ejecuta el microprocesador al encenderse es el de la BIOS
- general purpose registers

Bytes

| | | |
|----------|---|------|
| 2^{10} | K | kilo |
| 2^{20} | M | mega |
| 2^{30} | G | giga |
| 2^{40} | T | tera |
| 2^{50} | P | peta |
| 2^{60} | E | exa |

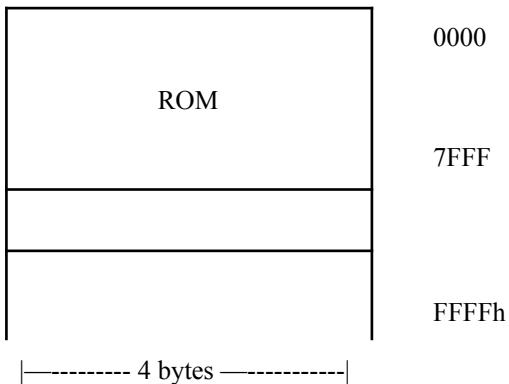
Mapa de memoria

- Supongamos un procesador que tiene 16 líneas de bus de direcciones y 8 líneas de bus de datos.
- $2^{16} = 2^6 K = 64K$ ROM
- $8/8 = 1$ byte de ancho → por el bus de datos



- ¿Y un procesador con 32 líneas de datos y 32 líneas de Direcciones ?

- $2^{32} = 2^2 G = 4G$ ROM
- $32/8 = 4$ bytes de ancho → dado por el bus de datos



Memorias

Ejemplo de pendrive:

- La lectura no vuelve a inyectar energía, pero si cuando se sobreescribe (no existe borrar, es pisar memoria).
- Con el tiempo puede llegar a perderse la memoria.
- La alineación a palabra del gcc aumenta la velocidad de lectura y escritura

- ROM:

- Read Only Memory
- **mantienen** su información SIN energía
- **la escritura es más lenta que la RAM**
- aca se almacena el BIOS → primer programa a ejecutarse
- Posiciones más altas en mapeo de memoria
- **PROM** (Programmable ROM) → firmware
- **EPROM** (Erasable PROM) → sistemas embebidos
- **Flash** → USB/ SSD
- **EEPROM** (Electric Erasable Programmable ROM) → BIOS

- RAM:

- Random Access Memory
- **pierde** su información SIN energía
- Posiciones más bajas en el mapeo de memoria
- **DRAM (Dinamic RAM)** → PC
- **SRAM (Static RAM)** → sistemas embebidos

- **Tiempo de accesos:** es el tiempo que le toma a una memoria RAM para completar un acceso después de otro

- **latencia:** tiempo que tarda en devolver el valor la memoria
- **transferencia**

- Operaciones:

- **acción a realizar** → lectura o escritura
- **dirección** de la palabra a acceder
- **dato** → entrante o saliente según acción
- **Estructura:** si el procesador, como el caso de Intel, quiere mantener compatibilidad hacia atrás, permite acceder a la memoria a nivel byte
 - la decodificación cambia el tipo de memoria

| | |
|-------|--|
| 0000h | |
| 0001h | |
| 0002h | |
| 0003h | |

Bus de 8 líneas

| | | | |
|-------|--|--|--|
| 0000h | | | |
| 0002h | | | |
| 0004h | | | |
| 0006h | | | |

Bus de 16 líneas

| | | | | |
|-------|--|--|--|--|
| 0000h | | | | |
| 0004h | | | | |
| 0008h | | | | |
| 000Ch | | | | |

Bus de 32 líneas

Integrados Compuertas y Decodificadores.....

Registros

- pequeña memoria dentro del procesador que cuando yo le caso la energía → desaparece
- tipo volátil → si no la alimentas, refrescas, etc se va
- A nivel eléctrico se resuelve con las instrucciones que mando en asm

NOR

| B | A | $\overline{B+A}$ |
|---|---|------------------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



| B | A | $B \oplus A$ |
|---|---|--------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



| B | A | $B \cdot A$ |
|---|---|-------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



| B | A | $B + A$ |
|---|---|---------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

BUFFER

| A | A |
|---|---|
| 0 | 0 |
| 1 | 1 |

NOT

| A | \overline{A} |
|---|----------------|
| 0 | 1 |
| 1 | 0 |



| B | A | $\overline{B \cdot A}$ |
|---|---|------------------------|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Compuertas

- nand → and negado
- xor → el o exclusivo
- buffer → como quieras

Decodificación de Hardware

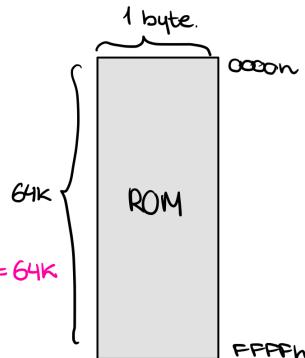
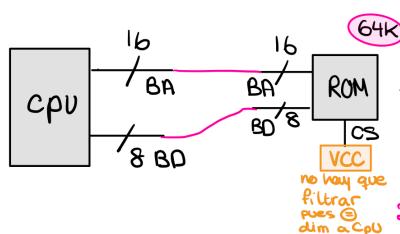
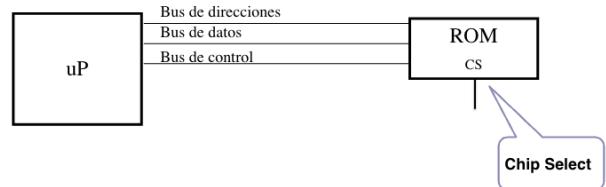
- Los decodificadores se emplean fundamentalmente para seleccionar los diferentes puertos de E/S (entrada/salida) y así la computadora pueda comunicarse con los diferentes dispositivos externos (periféricos)



ejercicio:

1.

- ¿Qué solución puedo implementar con este hardware?
- ¿Qué limitaciones tiene?



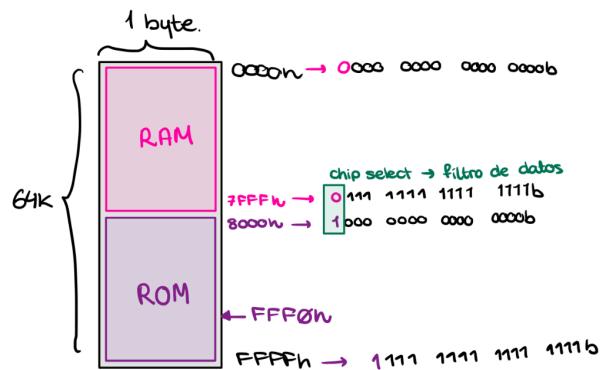
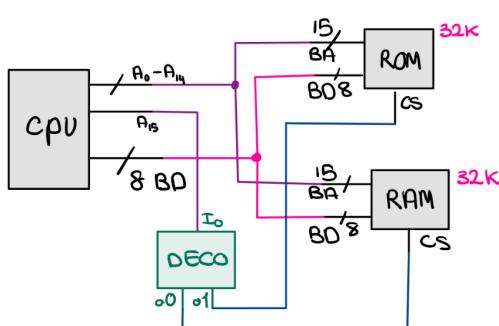
Limitación → no tengo RAM
↓ no tiene preferencias.

Podes → reloj digital → cuenta hrs, mins, y secs

mapa de memoria

$$32K = 2^5 \cdot 2^{10} = 2^{15} \text{ BA}$$

IP = FFFF0h \Leftarrow Bios va arriba \Rightarrow ROM va abajo.



Chip select

- le indica al periférico/RAM/ROM cuando el dato va por el bus de datos a esa dirección de memoria
- se usa el bus de address para activar el chip select

PROBLEMA: no podemos agregar un línea nueva en el bus de address porque no tenemos lugares de un bit más

SOLUCIÓN: crearon un patita nueva que se llama IO/M porque sino no había forma de activar el chip select de aquellos dispositivos pertenecientes al mapa paralelo de entrada y salida

IO/M patita

- indica a qué mapa se va ir el dispositivo
- IO/M = 1 va al mapa de entrada y salida → perifericos

```
IN [1000h]
OUT [1000h]
```

- IO/M = 0 va al mapa de memoria

```
MOV eax, [1000h]
```

- son Mapas paralelos que se usan para activar el chip select el bus de address

ej. bus de address tiene 16 bits

NO SE PUEDE AGREGAR UNA PATITA → porque ese bus esta totalmente relacionado con los registros entonces si vos agregas una patita más en el bus como escribis un dato en ese bus si no tenes un registro de 17 bits

R/^W patita

- esta línea indica si se va a leer un dato o se va escribir
- R/^W = 1 indica que el dispositivo activo debe **escribir** un dato en el bus de datos
- R/^W = 0 indica que debe **guardar** el dato que esté presente en el bus de datos

ejemplo: Como identificar en el código ASM

- escribir en memoria

```
mov eax, [1000h]
```

IO/M = 0 → el mov indica que va en el mapa de memoria
 R/^W = 1 → porque se escribe en el registro eax lo que contiene la posición 1000h en memoria

- guardar en memoria

```
mov eax, 1234h → dentro del procesador no afecta los buses
mov [1100h], eax → afecta el bus
```

IO/M = 0 → porque usa la instrucción mov
 R/^W = 0 → porque guarda lo del registro eax en la posición 1100h del mapa de memoria
 bus de datos: 1234h
 bus de address: 1100h

- mapa de E/S

```
IN ax, 70h
```

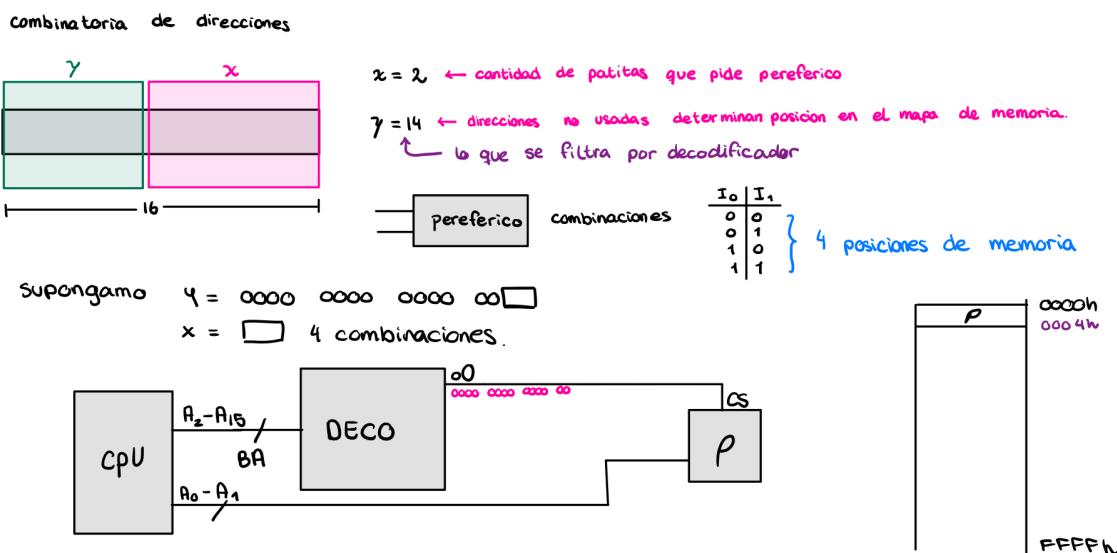
IO/M = 1 → porque usa la instrucción IN
 R/W = 1 → se escribe en el registro ax

bus de dirección: 70h
bus de datos: ??? lo que esté prestando el dispositivo

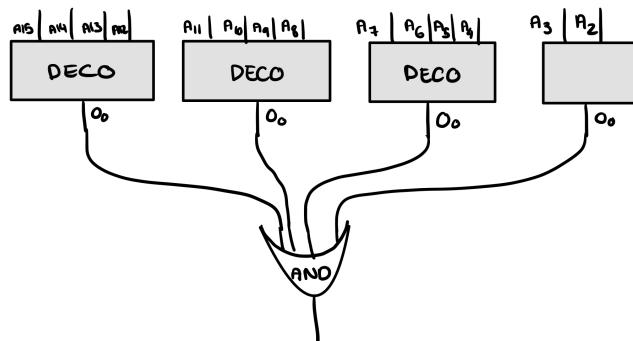
Periféricos

- Los periféricos son dispositivos externos conectados a una computadora u otro sistema informático que amplían sus capacidades o le permiten interactuar con el usuario.
- Estos dispositivos pueden ser de entrada, salida o ambos, y se utilizan para realizar tareas específicas o mejorar la experiencia del usuario.

ejemplo de cómo se mapea



el DECO en este caso estaría representando la cascada de decos que deberían ir ya que no sería factible crear un deco con 13 entradas ya que tendría 2^{13} salidas



Pines básicos de control

- **WR:** cuando vale cero hay escritura
- **RD:** cuando vale cero hay una lectura
- **IO/M:** si vale 1, **operaciones con ports**; si vale 0, **operaciones con la memoria**
 - El último se usa para conectar periféricos, se manda esa patita a un decodificador y la salida O1 es dirigida al periférico
 - O0 está dirigida al decodificador de las memorias.

| | | |
|-----------------|----|-----------------|
| X1 | 1 | V _{cc} |
| X2 | 2 | HOLD |
| RESET OUT | 3 | HLD |
| SOD | 4 | CLK(OUT) |
| SID | 5 | RESET IN' |
| TRAP | 6 | READY |
| RST7.5 | 7 | IO/M' |
| RST6.5 | 8 | S ₁ |
| RST5.5 | 9 | RD' |
| INTR | 10 | WR' |
| INTA' | 11 | ALE |
| AD ₀ | 12 | S ₀ |
| AD ₁ | 13 | A ₁₅ |
| AD ₂ | 14 | A ₁₄ |
| AD ₃ | 15 | A ₁₃ |
| AD ₄ | 16 | A ₁₂ |
| AD ₅ | 17 | A ₁₁ |
| AD ₆ | 18 | A ₁₀ |
| AD ₇ | 19 | A ₉ |
| V _{ss} | 20 | A ₈ |
| 8085A | | |

Decodificación de hardware

- **completa:** se dice que una decodificación es completa cuando hay una relación biunívoca entre cada posición de memoria y cada dirección
- **incompleta:** por simplicidad o para minimizar la cantidad de componentes no se hacen llegar todas las líneas del bus de direcciones

Sistemas de entrada y salida

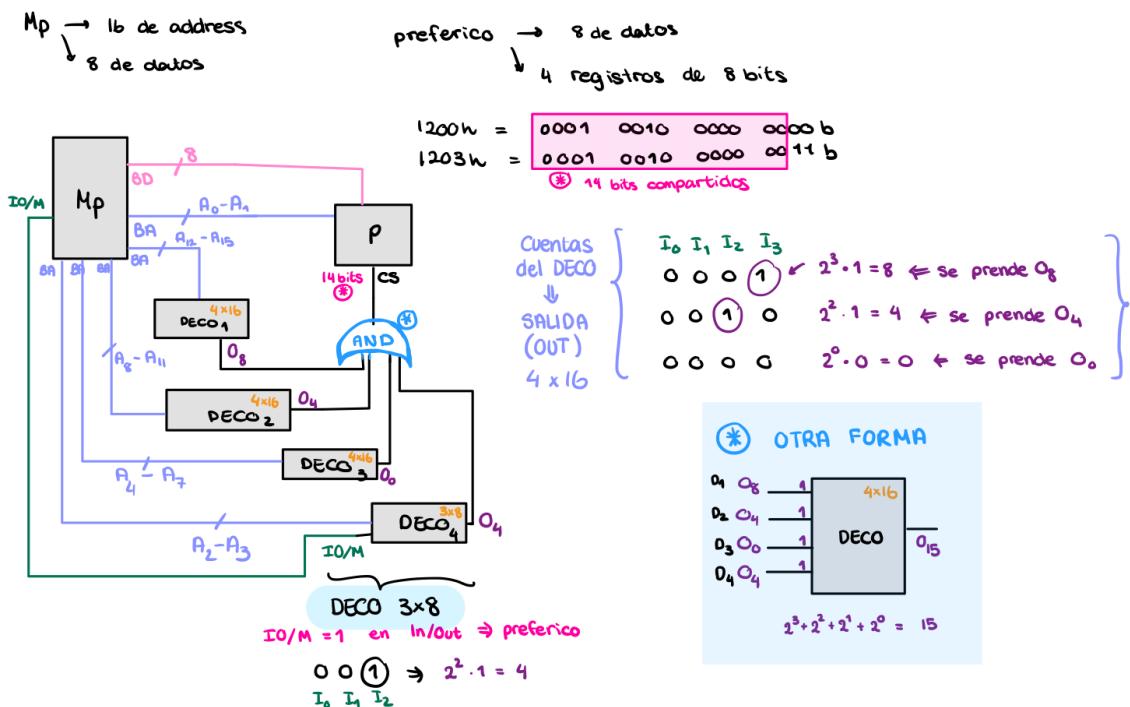
- **E/S aislada:** un señal externa interrumpe al micro indica la ejecución de una operación de E/S
- **interrupción:** una señal externa interrumpe al micro para requerir un servicio de atención
- **Acceso directo a memoria (DMA):** la información se transfiere directamente a la memoria, no requiere de intervención del CPU
 - pueden NO pasar por el procesador
 - ej. disco rígido → accede directo a la RAM y contiene la info con zonas de memoria pre-establecidas
- **mapeo de memoria:** se otorga un sector de memoria principal al dispositivo
 - aparecen:
 - RAM
 - ROM
 - periféricos

| instrucción | IO/M | R/W | Mapa | |
|---------------|------|-----|---------|----------------------|
| Mov al, 80h | 0 | 1 | memoria | registros RAM |
| mov [80h], ax | 0 | 0 | | ROM o RAM |
| in / out | 1 | | In/Out | periféricos |

TRUCO: Depende de que numero input I_n está prendido en uno sale por $\sum_{unos\ prendidos} 2^n = m$ será el output O_m

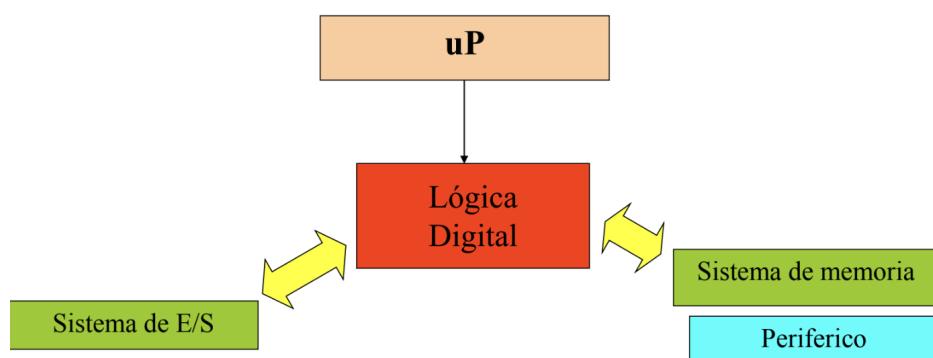
Inputs siendo de menos significativo a mayor SINO se hace de mas significativo y se hace la conversión del binario

Ejemplo:



Mapeo de memoria

- un dispositivo es manejado como una o varias posiciones de memoria
- cada posición de memoria puede ser un **registro interno diferente del dispositivo**
- VENTAJAS:
 - se utilizan la **instrucciones de acceso a memoria**, por lo tanto las alternativas de programación son mayores por tener mayor cantidad de instrucciones para manejo de memoria
 - se **modifica directamente los registros del periférico** sin necesidad de obtener el valor con las instrucciones IN y OUT
- DESVENTAJA:
 - **reduce cantidad de memoria**
 - el impacto es mínimo relacionado con la cantidad de memoria que tiene las PC actuales



Se decodifica al periférico en el mapa de memoria, cambiando la logica digital.

obs: siempre el deco va tener una patita en uno y el resto en cero

si vos a las dos salidas de un deco las unise, la unión va ser indefinida → NO se hace

Ejecución de una instrucción:

Cuando una instrucción debe ser ejecutada, primero debe ser traída desde la memoria. Este proceso se llama "fetch" de la instrucción, y es una parte del ciclo de instrucción que consta de varias etapas: fetch, decode, execute y, a veces, write back. El "fetch" implica acceder a la memoria para recuperar la instrucción que será ejecutada.

En términos de la operación de "fetch", esto generalmente se realiza de la siguiente manera:

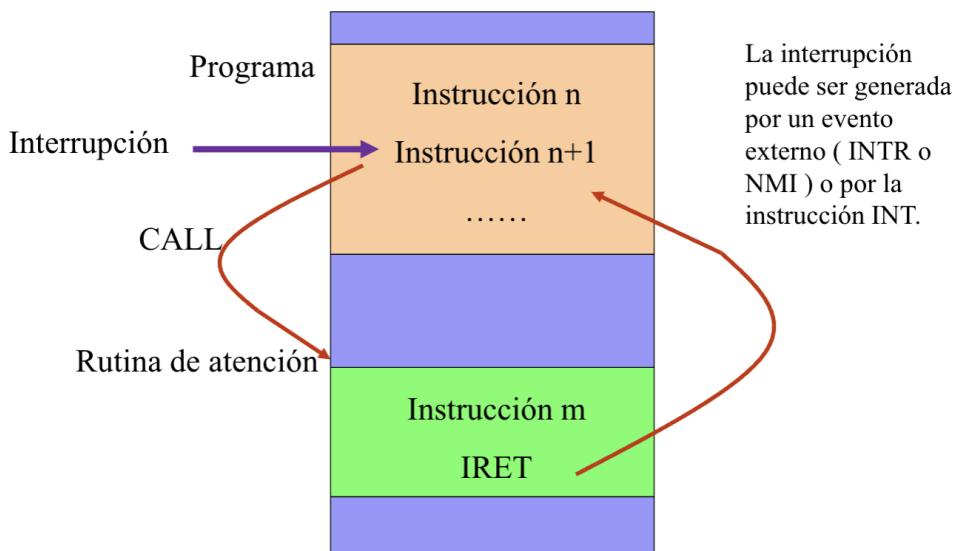
1. **Colocar la dirección de la instrucción en el bus de direcciones:** El registro contador de programa (PC) contiene la dirección de la siguiente instrucción a ejecutar. Esta dirección se coloca en el bus de direcciones.
2. **Señalar a la memoria para leer:** Se activa una línea de control que indica a la memoria que debe colocar el contenido de la dirección especificada en el bus de datos.
3. **Transferir la instrucción al registro de instrucciones (IR):** La instrucción recuperada de la memoria se transfiere al registro de instrucciones para su decodificación y ejecución.
4. **Incrementar el PC:** El contador de programa se incrementa para apuntar a la siguiente instrucción en la secuencia.

Interrupciones

Sistema de Entrada y Salida

- **Interrupción:** una señal externa interrumpe al microprocesador para requerir un servicio de atención
 - Es una **señal recibida por el procesador**, que indica que debe **interrumpir** el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.
- **E/S aislada:** una señal especial del micro indica la ejecución de una operación de E/S
- **Acceso directo a memoria(DMA):** la información se transfiere directamente a la memoria, no requiere de intervención del CPU
- **Mapeo en memoria:** se le otorga un sector de memoria principal al dispositivo

Rutina de atención de la interrupción (Drivers de interrupción)



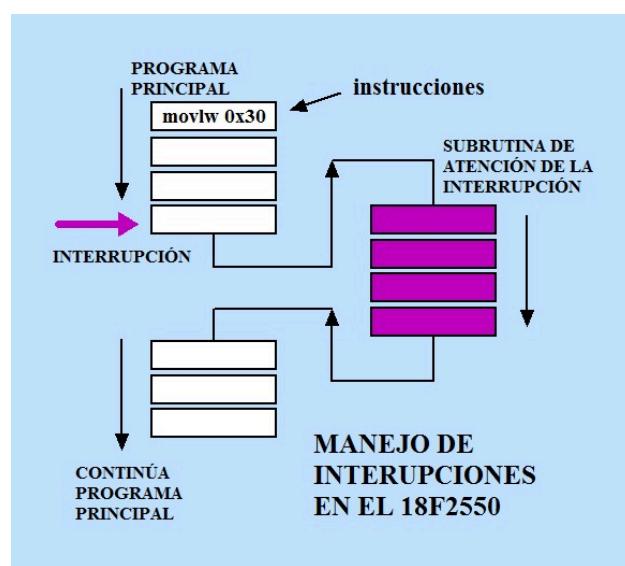
- se interrumpe el procesador para ejecutar una rutina de atención
- ej. la tecla
 - cuando se toca una tecla
 - se detiene el programa principal que se está corriendo y corre la rutina de atención de interrupción del teclado
 - cuando termina de correr vuelve al programa principal

OBS: nosotros trabajamos con **mono-procesadores** osea solo hay un procesador lo cual hace que haya **un solo IP** entonces solamente se pueden estar **ejecutando una instrucción** del procesador en **un instante de tiempo**

los núcleos no son distintos procesadores

“es como una cola en el super vos tenes 4 colas donde se van acumulando instrucciones y una salida del supermercado”

internamente se pueden ejecutar instrucciones **PERO** cuando las tenes que ir a buscar a memoria o cambiar un dato o hacer un incremento/decremento lo que sea **solamente sale de a una**



Tipos de interrupciones

- interrupciones de **hardware**:
 - se interrumpe la ejecución del programa activando alguna de las **dos entradas que tiene el microprocesador (INTR y NMI)**
 - **INTR → enmascarables**
 - no prestarle atención a las interrupciones si queres
 - poniendo el flag **IF=0**, que se controla con las instrucciones **sti** y **cli**
 - se usa ya que hay procesos muy **importantes** como escribir en el disco que **no pueden ser interrumpidos**
 - **procesos atómicos** → CUANDO ARRANCAN TERMINAN SI O SI

STI Set Interrupt
CLI Clear Interrupt

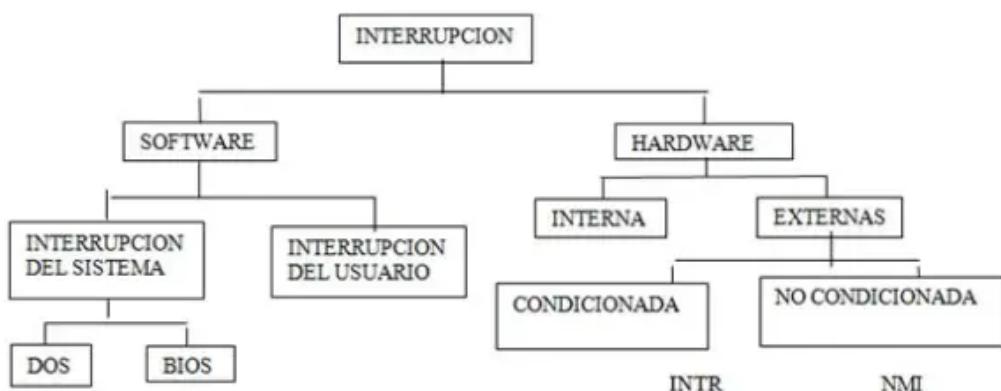
POPF → will not modify the **Interrupt flag**

- **NMI** → no se puede evitar → **no enmascarables**
 - **SI o SI interrumpe al sistema operativo**
 - Siempre se ejecuta la rutina que se encuentra en la **posición 2h** del vector de interrupciones(INT 2h).
 - Las únicas que no se pueden **bloquear** con: **sensor de temperatura, batería baja, apagado y alguna de ese nivel de importancia**

- interrupciones de **software**:

- se interrumpe la ejecución del programa **al ejecutar la instrucción de assembler INT**
- INT 44h siendo 44h el número de rutina de interrupción a ejecutar
- el número es la **posición en el vector de direcciones** y luego salta a esa rutina (IDT)

CALL es para llamar un función que hiciste vos
INT es para llamar la función que hizo otra persona → sistema operativo



PIC (Controlador programable de interrupciones)

- es un periférico
- como solo hay una patita para las interrupciones(INTR), el pic te permite crear más
 - **te da hasta 8 perifericos**
- **¿Qué hace?** pasa por el bus de datos que interrupcion es y la busca en el IDT y ejecuta la rutina
- los puertos de E/S del **PIC** son **20h y 21h**
 - 20h se usa para programar el PIC (lo que usa el BIOS)
 - 21h

in al, 21h //accede al pic, leo máscara del pic. Si 0 = máscara deshabilitada, pasa la señal al microprocesador.

mov al, 0FEh //habilito la interrupción de teclado

out 21h, al //así se programa el pic EOI end of interrupt, luego de ejecutar la rutina se avisa al pic que terminó la rutina

- Se puede especificar la **IRQ** que terminó o enviar un código que indica que finalizó la atención de la última interrupción que llegó.
 - Esa palabra se envía al puerto 20h y el valor que se envía para indicar que finalizó la atención de la interrupción es el valor 20h.
- PIC en cascada
 - colocando 2 PICs en cascada se amplía la cantidad de interrupciones de hardware en la PC
 - IRQ2 como

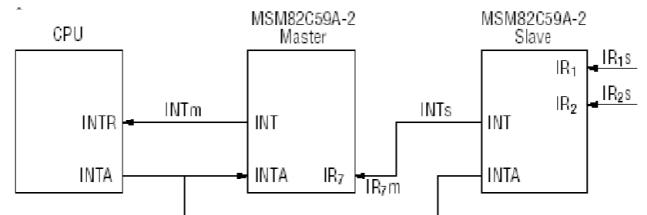


Fig. 1 System Configuration

| Línea IRQ | INT Tipo | Descripción |
|-----------|----------|-------------------------------------|
| IRQ0 | 08h | Timmer tick (18.2 veces por seg.) |
| IRQ1 | 09h | Teclado |
| IRQ2 | 0Ah | INT desde 8259A esclavo |
| IRQ8 | 70h | Servicio de reloj en tiempo real. |
| IRQ9 | 71h | Redireccionamiento por soft. a IRQ2 |
| IRQ10 | 72h | Reservada |
| IRQ11 | 73h | Reservada |
| IRQ12 | 74h | Reservada. |
| IRQ13 | 75h | Coprocesador numérico. |
| IRQ14 | 76h | Controlador de disco rígido. |
| IRQ15 | 77h | Reservada. |
| IRQ3 | 0Bh | COM2 |
| IRQ4 | 0Ch | COM1 |
| IRQ5 | 0Dh | LPT2 |
| IRQ6 | 0Eh | FLOPPY |
| IRQ7 | 0Fh | LPT1 |

| | |
|--|---------------------|
| (línea IRQ) → 16 patitas de los dos PIC (INT tipo) → 16 entradas de la IDT | esta pasa en la RAM |
|--|---------------------|

| | |
|--|--|
| (descripción) → programado por default | |
|--|--|

BIOS

- el bios al iniciar la PC guarda en memoria rutinas básicas para poder empezar a operar

| | |
|---------|----------------------------------|
| INT 10h | Rutinas de video (BIOS) |
| INT 13h | Rutinas de disco (BIOS) |
| INT 14h | Rutinas para puerto Serie (BIOS) |
| INT 19h | Rutina para bootloader (BIOS) |
| INT 1Ah | Rutinas para el RTC (BIOS) |

- **en la ROM**
- **no volátil**
- se corre cuando se prende la maquina y luego no se usan más hasta arrancar la compu devuelta
 - una vez que se corri al inicio luego se loadea la IDT y se corren los punteros de función de la IDT

IRQ (Interrupt Request Queue)

- Los dispositivos se comunican con el procesador mediante líneas de interrupción únicas llamadas **IRQ**.
- Las **IRQ** son gestionadas por **el controlador de interrupciones**, que puede estar integrado en el procesador o ser un circuito separado.
- **El controlador de interrupciones:**
 - **Habilita o inhibe** las líneas de interrupción.
 - **Establece prioridades** entre las interrupciones.

Si varias IRQ se activan a la vez, **el controlador elige cuál informar** al procesador basándose en prioridades.

- Un procesador **sin controlador de interrupciones** integrado usa una línea de interrupción llamada **INT**.
 - Cuando se activa la línea **INT**
 - el procesador consulta el controlador de interrupciones para **identificar la IRQ a atender**.
 - El procesador busca en **la tabla de vectores de interrupción (IDT)** la rutina correspondiente para **manejar la solicitud del dispositivo asociado a esa IRQ**.

Excepciones

- funciona **EXACTAMENTE igual** a las interrupciones **PERO se generan por el mismo procesador**, osea el procesador se interrumpe a sí mismo

- es un evento generado por el procesador cuando detecta una o más condiciones predefinidas al ejecutar una instrucción

TIPOS:

- **faults**: se pueden **corregir**, se guarda la dirección de donde viene la falla
- **trap**: se utilizan para **acceder** al SO desde un programa del usuario
- **abort**: **no** se puede **corregir**

- hay 32 (ocupan los primeros 32 lugares de la IDT)
 - por defecto la división por 0
 - 1. detección del error: cuando un programa intenta realizar la división por 0
 - 2. generación de la interrupción: el procesador genera una excepción o interrupción específica llamada "divide error"
 - a. detiene la ejecución normal del programa y pasa el control a un mecanismo de manejo de excepciones
 - 3. mensaje de error

| Id | Description |
|--------|--|
| 0 | Divide error |
| 1 | Debug exceptions |
| 2 | Nonmaskable interrupt |
| 3 | Breakpoint (one-byte INT 3 instruction) |
| 4 | Overflow (INTO instruction) |
| 5 | Bounds check (BOUND instruction) |
| 6 | Invalid opcode |
| 7 | Coprocessor not available |
| 8 | Double fault |
| 9 | (reserved) |
| 10 | Invalid TSS |
| 11 | Segment not present |
| 12 | Stack exception |
| 13 | General protection |
| 14 | Page fault |
| 15 | (reserved) |
| 16 | Coprocessor error |
| 17-31 | (reserved) |
| 32-255 | Available for external interrupts via INTR pin |

El procesador detecta que hay un error en tu programación

El procesador maneja directamente la ejecución del programa y solo involucra al sistema operativo cuando se produce una interrupción o excepción.

¿Porque no lo detecta por SO?

Porque el **SO no está corriendo** mientras corres tu programa, entonces **no hay chance** que el SO detecte el error NO ESTÁ CORRIENDO

El procesador **cuando detecta un error se auto interrumpe** y llama al SO corriendo una rutina creada por el SO para que venga a arreglar lo que hizo mal el usuario

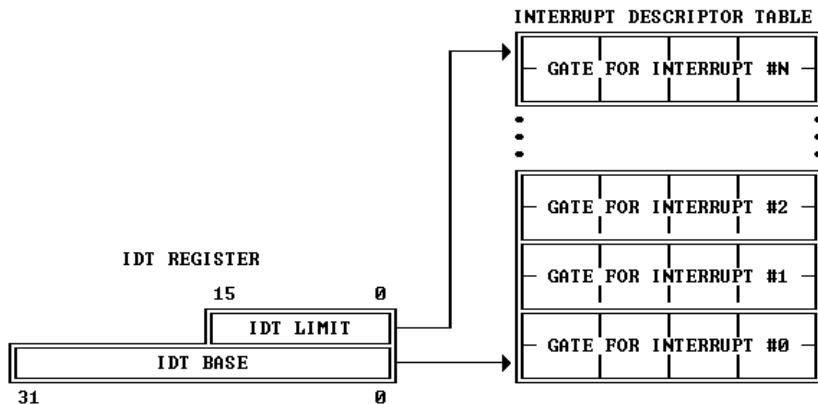
EN RESUMEN:

Cuando ocurre un **error en la ejecución de un programa**, el **procesador lo detecta, se interrumpe** y llama al sistema operativo mediante una interrupción o excepción. El sistema operativo, entonces, **maneja el error a través de sus rutinas predefinidas**.

IDT Loader

- **en la RAM**
- Es responsable de **configurar** la tabla de **descriptores de interrupción**, la cual define cómo el procesador **maneja las interrupciones y excepciones**.

Figure 9-1. IDT Register and Table



cómo funciona:

1. Definir las estructuras del IDT y del registro IDTR.
2. Configurar cada entrada del IDT con las direcciones de las rutinas de servicio de interrupción.
3. Cargar el IDT usando la instrucción lidt

TPE - explicado

Estructura

| Kernel Space | | User Space |
|-----------------------|---|------------------------------------|
| Keyboard Driver | ← Syscalls es la forma → de comunicarse | Shell |
| Timer Driver | están separados para limitar y gestionar el acceso | funcionales pedidas por cátedra |
| Video Driver | a los distintos recursos y periféricos de la | |
| Interrupciones | computadora por parte del usuario → pues es peligroso | |
| Manejo de excepciones | que el usuario tenga acceso directo a estos recursos protegidos | |

Kernel Space

- interactúa directamente con hardware, mediante drivers, y al mismo tiempo provee funciones al User Space
- abstraer al usuario y brindarle un determinado conjunto de funciones, controladas mediante drivers, para acceder a los recursos de manera segura

Keyboard Driver

- **Al atender una interrupción:**
 - Recibe el scan code de la tecla presionada.
 - Si no es una tecla especial, guarda el carácter ASCII correspondiente en un buffer local del driver.

- **Mediante una syscall:**
 - Se puede obtener lo almacenado en el buffer local del driver.
 - La syscall recibe dos parámetros:
 - La longitud que se pretende leer del buffer.
 - Un vector (o array) donde se guardará el contenido leído.
 - La syscall devuelve:
 - La cantidad de caracteres leídos del buffer.
 - 0 si no se leyó nada.
- **Desventaja de este diseño:**
 - Costoso en ejecución, ya que para leer un carácter implica realizar numerosas syscalls constantemente hasta que el usuario ingrese algo.
- **Para el TPE:**
 - Se deben definir teclas para generar un snapshot de los registros en el momento de la ejecución.
 - Al ejecutarse la rutina de atención de interrupciones, se pushean los registros.
 - Con la dirección del rsp (Stack Pointer), que se obtiene como argumento de la función, se puede realizar el snapshot.

Timer Driver

- de modo de poder proveer al usuario con la fecha y horario local del sistema de formato amigable
- RTC
- crear un syscall que lo devuelva de forma tipo date

Video Driver

- **Manejo de píxeles de la pantalla:**
 - Provee funciones para graficar píxeles simples, matrices de píxeles y caracteres.
 - El driver provee:
 - Un sistema de representación de caracteres.
 - El cursor de la posición actual del usuario en la pantalla.
 - Todas las acciones del usuario en la pantalla están controladas por el driver.
 - A pesar de la libertad para elegir dónde graficar (matrices o caracteres), existen limitaciones para un uso seguro.
- **Librería adicional:**
 - Genera una leve abstracción del driver.
 - Añade el concepto de string o cadena de caracteres.
 - Permite imprimir múltiples caracteres seguidos, reduciendo la cantidad de interrupciones de software, lo que optimiza la ejecución.
- **Funciones adicionales:**
 - Proveen datos de la pantalla como el ancho y alto en píxeles (resolución).
 - Estos datos son fundamentales para que el usuario pueda diseñar interfaces.

Interrupciones

- **Distinción entre interrupciones:**
 - Separación entre interrupciones generadas por software, hardware y excepciones.
 - Permite un manejo más prolífico y uniforme de las interrupciones.

- **Organización en tres archivos distintos:**
 - `irqDispatcher`: Encargado de las interrupciones de hardware.
 - `syscallDispatcher`: Encargado de las interrupciones generadas por software (int_80h).
 - Archivo de manejo de excepciones: Encargado de las excepciones.
- **Beneficios de esta organización:**
 - Agrupación de interrupciones en base a sus comportamientos y parámetros necesarios para su correcto funcionamiento.
 - Mejora en la estructura y mantenimiento del código.

Excepciones

- Preservar los valores iniciales de los registros `rip` y `rsp`:
 - `rip`: Para tener una próxima instrucción a ejecutar.
 - `rsp`: Para dejar el stack en su posición inicial, considerando su crecimiento.
- Se guarda el valor de `sampleCodeModuleAddress` y `rsp` antes de inicializar la shell por primera vez.
- **Al ocurrir una excepción**, se puede restaurar el flujo habitual del programa:
 - Asignando nuevamente sus valores iniciales a los registros `rip` y `rsp`.
- **Se modifican los valores del stack frame** que se pushean al atender una interrupción.
- Esto aprovecha el hecho de que una **excepción es un tipo de interrupción**.

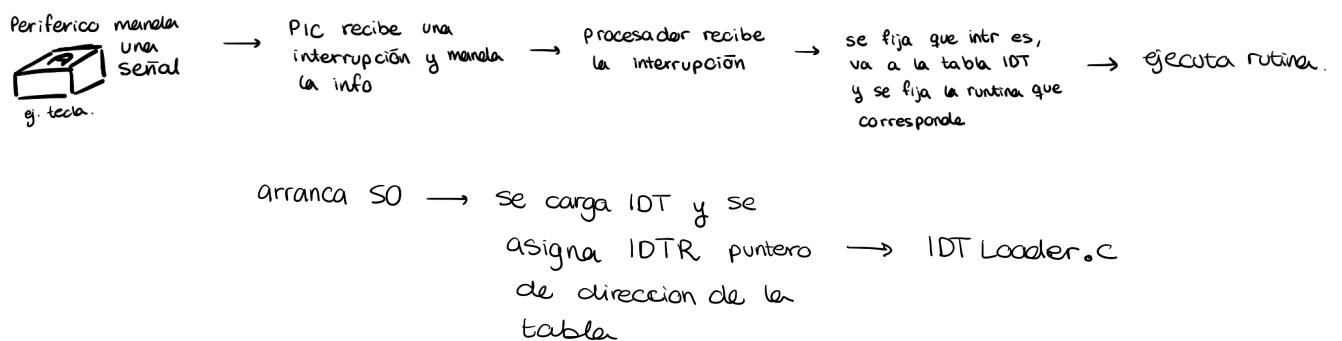
User Space

- contiene funcionalidad provistas para un usuario
- **Interacción del usuario con funciones:**
 - El usuario interactúa directamente con distintas aplicaciones o librerías que facilitan el acceso a diversas funcionalidades implementadas.
 - Estas funciones no deben acceder directamente a recursos protegidos. En su lugar, deben utilizar interrupciones de software específicas para solicitar datos o servicios.
- **Uso de librerías para modularización:**
 - Para mejorar la modularización del código, se han desarrollado librerías que encapsulan casos particulares de syscalls genéricas.
 - Por ejemplo, la función `println` incluida en la librería `string` se encarga de imprimir texto junto con un salto de línea utilizando únicamente la syscall `write`.

Shell

- **Funcionamiento de la Shell:**
 - Es un intérprete de comandos que constantemente lee del buffer de teclado mediante una syscall.
 - Permanece en un bucle leyendo hasta recibir un comando válido.
 - Cuando recibe un comando válido, ejecuta su aplicación correspondiente y luego vuelve a la shell.
- **Acceso a recursos como la pantalla:**
 - Siguiendo los principios explicados en la sección del driver de Video, el acceso indiscriminado a recursos como la pantalla es peligroso.

- La Shell utiliza las funciones estándar provistas por las syscalls para interactuar con la pantalla.
- No decide directamente dónde se imprime en la pantalla; esto se delega al Kernel.
- **Interfaz de usuario:**
 - Para generar una interfaz más intuitiva y compatible con otras implementaciones de terminales:
 - A medida que el usuario ingresa caracteres para insertar un comando, estos se van mostrando en pantalla.
 - Esto ayuda a proporcionar retroalimentación visual al usuario durante la entrada del comando.



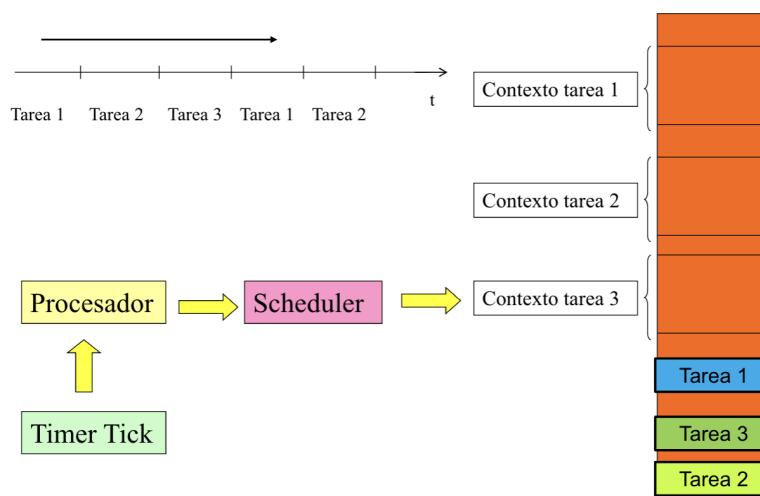
Modo Protegido.....

¿Qué es?

- es una forma de proteger a los programas del sistema para usar funcionalidades como memoria virtual, paginación, y hacer multitareas
- está diseñado para mejorar el control del SO sobre las aplicaciones del programa
- NO LO HACE EL SO → pues si no esta corriendo no lo puede proteger
- Lo hace el microprocesador
 - crea tablas para proteger

Comunicación de tareas

- Va intercalando la ejecución de las tareas, es tan rápido que parece que se ejecutan en **simultáneo**. Puede ejecutar en simultáneo pero es multi-core. Dentro de un core se hace esto.



Timer tick:

cada x tiempo (por default son 55 milisegundos) **interrumpe** al procesador y ahí se llama al scheduler. Es un **periférico que genera interrupciones mediante el pic.**

NUNCA PARA

Scheduler:

La rutina **define** qué tarea se ejecuta luego de **cada interrupción del timer tick**, guarda el contexto y recupera el contexto de la próxima tarea.

- Se le puede **dar prioridad** a alguna tarea en **específico**, le da **más tiempo** de procesador a la que tenga más prioridad.
- Se usa el modo protegido **para que una tarea no pueda romper a otra** y se pueda hacer **multitarea** (protección de tareas).
- El contexto de la tarea **guarda el valor de los registros y stack** de cada tarea, porque cuando se ejecute otra cosa **va a romper los registros, se guardan en RAM**. Se ocupa del scheduler.
- **El SO también es una tarea que funciona como cualquier otra.** Tiene prioridad sobre el resto, cada vez que **interrumpe el timer tick se llama a él**. El scheduler es parte del SO.

El SO no puede evitar que una tarea acceda a otra, porque **es una tarea que no corre mientras hay otra corriendo.**

El que evita que esto suceda es el procesador. Verifica con unas tablas guardadas en memoria con la información de a que puede acceder cada cosa. Se lanza una excepción (ej segfault).

¿Que pasa cuando **no hay que te pida el procesador**, a quien le entregas el procesador si no hay nadie?

IP (apunta a la próxima instrucción a correrse) → **siempre tiene correr algo**

SIEMPRE TIENE QUE CORRER ALGO → los SO **siempre** tienen un proceso, que no hace nada, para entregarle el procesador cuando no hay nadie que lo quiera
en LINUX → idle process

ej.

en WINDOWS → proceso inactivo del sistema

- PID=0
- 90 algo del CPU → es la cantidad de veces sobre 100 que le diste al procesador el proceso inactivo del sistema
- 8K de memoria
- es un while(1) no te consume procesamiento, consume tiempo no más

| Task Manager | | | | | | |
|---------------------|-------|---------|-----------|------|---------------|----------------------------------|
| Name | PID | Status | User name | CPUs | Memory (a...) | Description |
| System Idle Process | 0 | Running | SYSTEM | 96 | 8 K | Percentage of time the processor |
| chrome.exe | 20412 | Running | chris | 01 | 80,288 K | Google Chrome |
| System interrupts | - | Running | SYSTEM | 01 | 0 K | Deferred procedure calls and |
| dwm.exe | 1160 | Running | DWM-1 | 01 | 23,344 K | Desktop Window Manager |
| chrome.exe | 18280 | Running | chris | 01 | 377,360 K | Google Chrome |
| Taskmgr.exe | 5980 | Running | chris | 00 | 30,008 K | Task Manager |

analogía no computacional “vos podes alquilar un auto por 7 dias y no hacerle kilometros”

EN RESUMEN Pasos:**Cada llamada de timer tick**

1. llega una interrupción
2. se detiene la tarea que estaba ejecutando

Scheduler

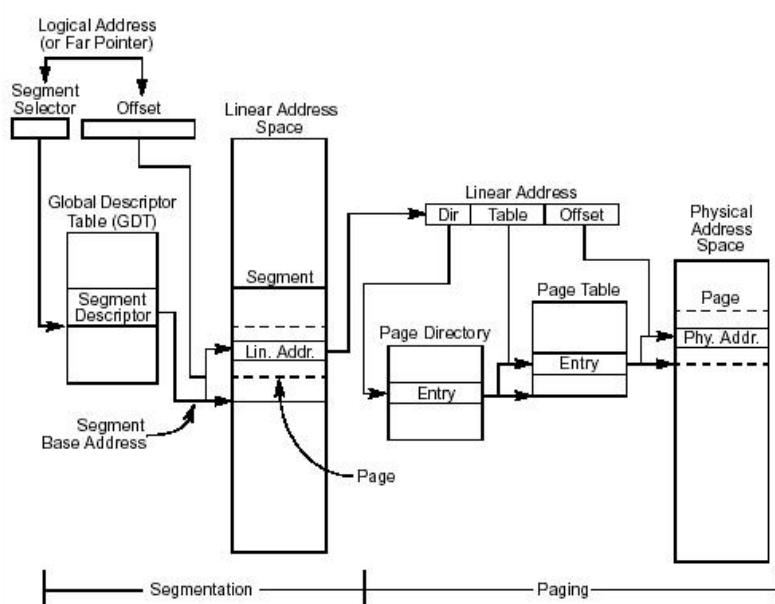
3. se backupea el contexto de la tarea en memoria (TODOS LOS REGISTROS DE LAS TAREA)
4. se decide cual es la próxima tarea a ejecutar
5. recuperas el contexto de la tarea elegida
6. se empieza a correr

SO es una tarea más → TIENE MAYOR PRIORIDAD

- le toca cuando hay un error o excepciones, cuando hay un interrupción, y cuando no corre ningun programa
- **no tiene las mismas reglas de quien le toca correr** o no porq el es quien decide cual es la próxima tarea en ejecutarse

Protección de tareas → MMU memory management unit

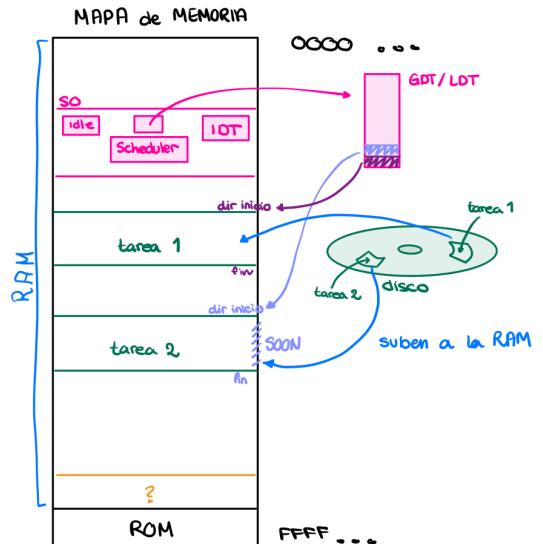
- el procesador controla las instrucciones
- ¿cómo lo hace?
 - vos le tenes que dejar una tablas que le indican cómo funcionar
- **Unidad de Segmentación:** divide la memoria en partes de tamaño **variable**
 - **NO se puede deshabilitar**
- **Unidad de Paginación:** divide la memoria en tamaño **fijo**
 - Si se puede deshabilitar
- **Memoria Física → RAM o ROM**
 - no siempre la dirección lógica coincide con la dirección física porque puede sufrir alteraciones
 - cuando se te acaba la memoria llevas a los programas al disco
- Unidad de segmentación y paginación se les asigna un espacio de memoria **VIRTUAL**



Debido a esta limitación (**no poder deshabilitar la segmentación**) de los procesadores Intel, algunos sistemas operativos **utilizan la memoria “Flat”** → **crear una sola pagina de segmentación que abarca todo**

Descriptor Table

- Segmentación:
 - **GDT(Global descriptor table)**: cada elemento de la tabla es un descriptor
 - descriptor → describe un segmento de memoria
 - **LDT(local descriptor table)**
- **IDT(Interrupt descriptor table)**



Descriptores

| 31 | | | | | | | | 16 | | | | | | | | 0 | | | | | | | | | | | | | | | | | |
|---------------------|--|--|--|---------------|--|--|--|----------------------|-----|------|--|--|------|-----------------------------|--|---|--|-----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| SEGMENT BASE 15...0 | | | | | | | | SEGMENT LIMIT 15...0 | | | | | | | | 0 | | | | | | | | | | | | | | | | | |
| BASE 31...24 | | | | LIMIT 19...16 | | | | P | DPL | TYPE | | | | BASE 23...16 | | | | + 4 | | | | | | | | | | | | | | | |
| Type | Defines | | | | | | | | | | | | Type | Defines | | | | | | | | | | | | | | | | | | | |
| 0 | Invalid | | | | | | | | | | | | 8 | Invalid | | | | | | | | | | | | | | | | | | | |
| 1 | Available 80286 TSS | | | | | | | | | | | | 9 | Available Intel386™ DX TSS | | | | | | | | | | | | | | | | | | | |
| 2 | LDT | | | | | | | | | | | | A | Undefined (Intel Reserved) | | | | | | | | | | | | | | | | | | | |
| 3 | Busy 80286 TSS | | | | | | | | | | | | B | Busy Intel386™ DX TSS | | | | | | | | | | | | | | | | | | | |
| 4 | 80286 Call Gate | | | | | | | | | | | | C | Intel386™ DX Call Gate | | | | | | | | | | | | | | | | | | | |
| 5 | Task Gate (for 80286 or Intel386™ DX Task) | | | | | | | | | | | | D | Undefined (Intel Reserved) | | | | | | | | | | | | | | | | | | | |
| 6 | 80286 Interrupt Gate | | | | | | | | | | | | E | Intel386™ DX Interrupt Gate | | | | | | | | | | | | | | | | | | | |
| 7 | 80286 Trap Gate | | | | | | | | | | | | F | Intel386™ DX Trap Gate | | | | | | | | | | | | | | | | | | | |

NOTE:

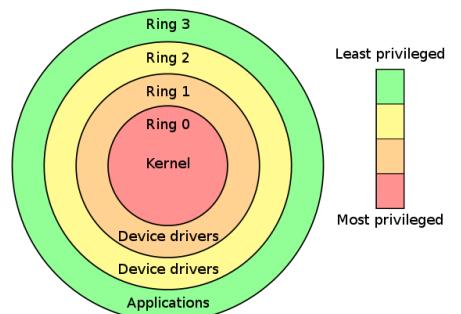
In a maximum-size segment (ie, a segment with G=1 and segment limit 19...0=FFFFFH), the lowest 12 bits of the segment base should be zero (ie, segment base 11...000=000H).

Segment limit:

- **P**: lo primero que se chequea es 'P' va indicar si existe físicamente o no
 - Si P=1 el segmento **está cargado** en la memoria física.
 - Si P=0 es segmento **está ausente**
 - cuando vas a buscar un segmento en memoria y está marcado **P=0** va lanzar **un excepción de falta de segmento** y va correr una rutina va ir a buscar donde esta ese segmento porque lo están pidiendo
 - cuando hay un segmento que ahora lo llaman y tiene que cambiar P=0 tiene que cambiar a donde apunta porque no puede haber dos programas con P=1 apuntando al mismo segmento

- **DPL** (descriptor privilege level): indica el nivel de privilegio del segmento

- Hay 4 niveles pero solo se usan 2 que son **kernel space y user space**.
- Si yo estoy en user space no voy a poder acceder a kernel.
- 0 maximo y 3 minimo → ver nivel de privilegio



- **S tipo de segmento**

- Si S=1 se trata de un segmento normal (código datos o pila)
- Si S=0 se trata de un segmento de sistema (puerta de llamada, TSS, etc)

- **TYPE:** (3 bits)

| | | | |
|-----|---|---|-----------------|
| E=1 | C | R | segmento normal |
|-----|---|---|-----------------|

| | | | |
|-----|----|---|-------------------|
| E=0 | ED | R | segmento de datos |
|-----|----|---|-------------------|

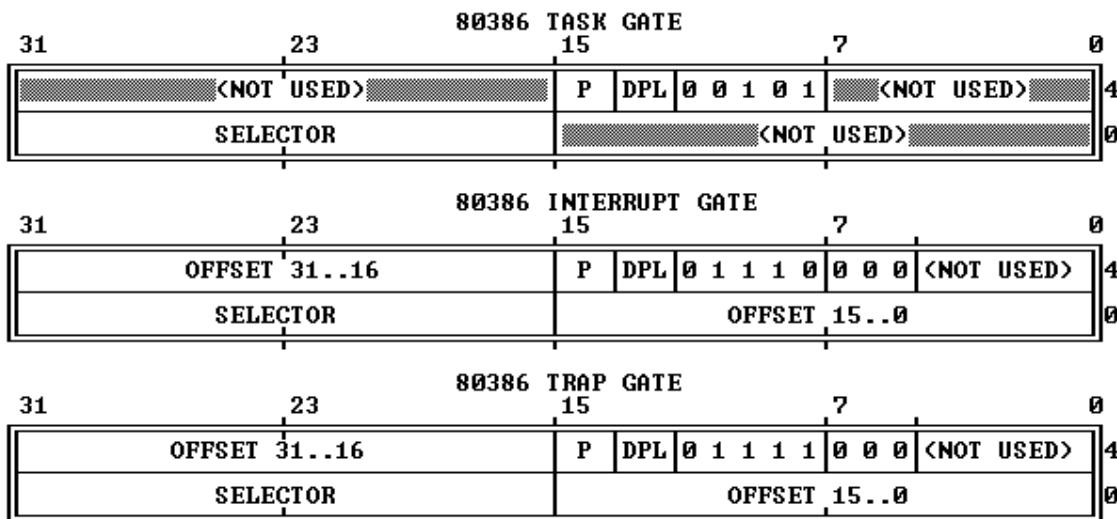
- E: ejecutable
 - E=1 segmento ejecutable
 - E=0 segmento de datos
- C: ajustable o conforming
 - C=1 cambio de nivel de privilegio del segmento
 - C=0 no cambio
- ED: Expansión decreciente
 - ED=0 un segmento de datos normal
 - ED=1 segmento de pila
- R: Read
 - R=1 leible
 - R=0 no legible
- W: Write
 - W=1 escribible
 - W=0 solo lectura

- **Tipos cuando S=0 se lo denomina ‘descriptor de sistem’ → los de la IDT**

- **trap gate (puerta de excepciones)**
 - Función: Maneja excepciones y ciertas interrupciones sin desactivar otras interrupciones.
 - **Uso: Excepciones** (errores de página, desbordamientos, etc.).
 - Estructura: Similar a un Interrupt Gate, pero no desactiva las interrupciones.
- **interrupt gate (puerta de interrupción)**
 - Función: Maneja interrupciones de hardware, desactivando interrupciones al entrar en el manejador.
 - **Uso: Interrupciones de hardware (temporizadores, teclados, etc.).**
 - Estructura: Incluye la dirección del manejador y un selector de segmento de código.

- **task gate**

- Función: Realiza un cambio completo de tarea utilizando la TSS (Task State Segment).
- **Uso: Multitarea basada en hardware.**
- Estructura: Contiene un selector que apunta a una TSS en la GDT

Figure 9-3. 80306 IDT Gate Descriptors

Dirección lineal → Memoria Virtual

ejemplo:

DS = 20h offset = 1000h

instrucción: mov [1000h], EAX ; default DS.

20h =

| | | | | |
|------|------|------|------|---|
| 0000 | 0000 | 0010 | 0000 | b |
|------|------|------|------|---|

request privilege level
↑
compara nivel de privilegio entre unos y otros
entrada en GDT
nro. 4
GDT=0
LDT=1

nro 4
GDT
5600 0000h ← dirección lineal
ahora con instrucción
mov [1000h], EAX
me muevo 1000h de la
posición relativa dentro del
segmento que te dieron
dentro de la GDT
5600 1000h → dirección real/física
que vas a acceder
en memoria

P = 0 en DS:20h no está en memoria
viene un nuevo programa DS:30h xq no
puede tener el mismo al anterior

30h =

| | | | |
|------|------|------|-------|
| 0000 | 0000 | 0011 | 0000h |
|------|------|------|-------|

nro 6.

GDT
nro 4
5600 0000h
P=0 ← cambia para no
existir en memoria
física
nro 6
5600 0000h
P=1

Vuelve programa viejo no puede
usar mismo lugar de mem.
física ya está ocupada
busca uno que esté libre
ej. 8800 0000h

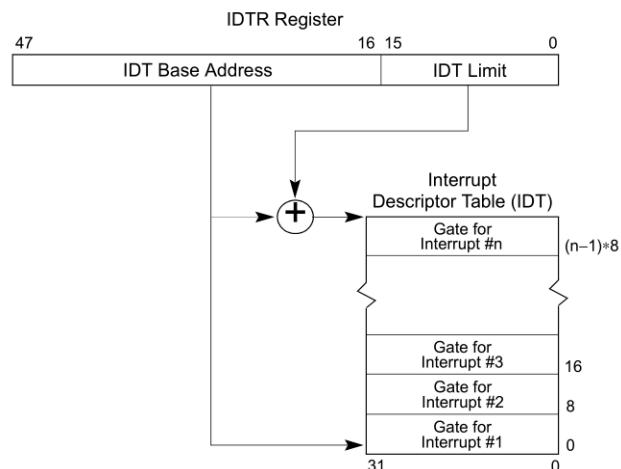
GDT
nro 4
8800 0000h
P=1 ← cambia bit
de presente en descriptor
porque ahora
presente en memoria física
nro 6
5600 0000h
P=1

Lo que cambia es la dirección base NO el Data Segment (DS)

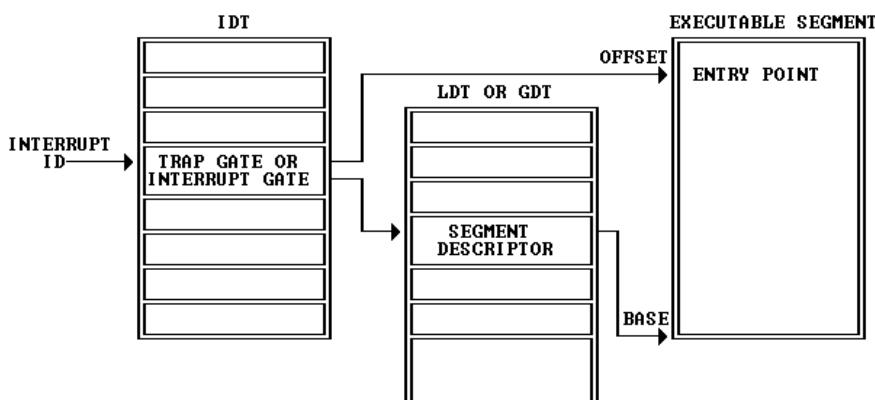
Cuando se apaga la computadora desaparecen todos los DS y los programas se guardan en disco

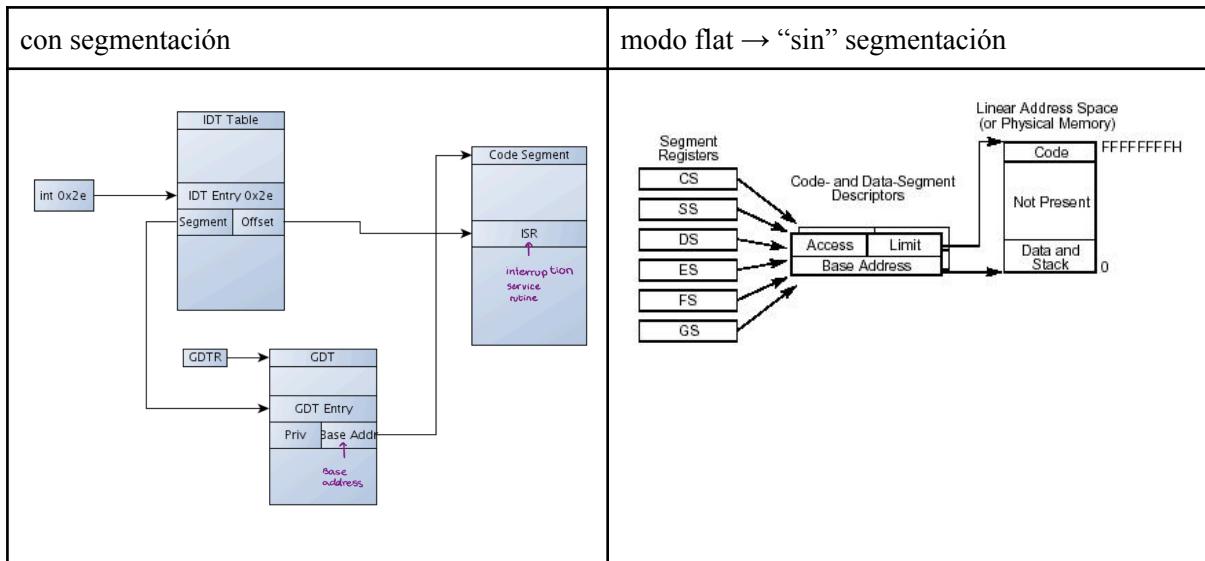
Carga de IDT

- Definir la estructura de la IDT:
 - Primero, define la estructura de una entrada de la IDT y el puntero de la IDT. Cada entrada describe una interrupción o excepción, incluyendo la dirección del manejador y atributos específicos. El puntero de la IDT contiene la base y el límite de la tabla.



- Crear la tabla IDT:
 - Declara un arreglo para las entradas de la IDT y una variable para el puntero de la IDT. El arreglo almacenará todas las entradas de la IDT y el puntero contendrá la información sobre la ubicación y tamaño de la IDT.
- Configurar una entrada en la IDT:
 - Define una función que establece los valores en cada entrada de la IDT, incluyendo la dirección del manejador de interrupciones, el selector de segmento y los atributos de la entrada. Asegúrate de asignar correctamente los valores de dirección y atributos para que el procesador pueda manejar las interrupciones de manera adecuada
- Cargar la IDT con lidt:
 - Configura el puntero de la IDT con la base y el límite de la tabla y utiliza la instrucción lidt para cargar la IDT en el CPU. Esto informa al procesador dónde encontrar la tabla de descriptores de interrupciones.
- Inicializar y configurar la IDT:
 - Inicializa las entradas de la IDT con las direcciones de los manejadores de interrupciones. Luego, llama a la función que carga la IDT (usualmente lidt). Este paso asegura que todas las posibles interrupciones están correctamente manejadas por el sistema operativo.
- Implementar los manejadores de interrupciones:
 - Implementa los manejadores de interrupciones en ensamblador y/o en C. Estos manejadores son las rutinas que se ejecutarán cuando ocurra una interrupción. Asegúrate de que los manejadores guardan y restauran el estado del procesador adecuadamente y manejen la interrupción según sea necesario.





Interrupciones en modo protegido

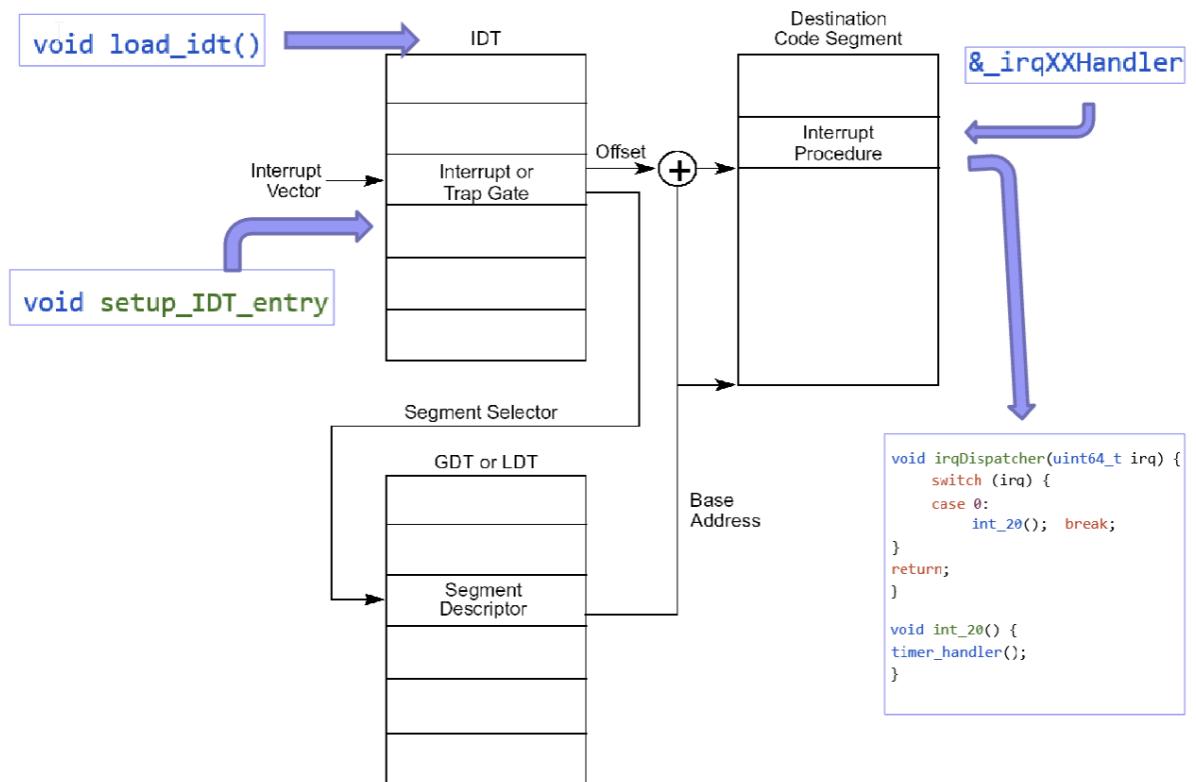


Tabla de excepciones

| Vector No. | Mnemonic | Description | Source |
|------------|----------|--|---|
| 0 | #DE | Divide Error | DIV and IDIV instructions. |
| 1 | #DB | Debug | Any code or data reference. |
| 2 | | NMI Interrupt | Non-maskable external interrupt. |
| 3 | #BP | Breakpoint | INT 3 instruction. |
| 4 | #OF | Overflow | INTO instruction. |
| 5 | #BR | BOUND Range Exceeded | BOUND instruction. |
| 6 | #UD | Invalid Opcode (UnDefined Opcode) | UD2 instruction or reserved opcode. ¹ |
| 7 | #NM | Device Not Available (No Math Coprocessor) | Floating-point or WAIT/FWAIT instruction. |
| 8 | #DF | Double Fault | Any instruction that can generate an exception, an NMI, or an INTR. |
| 9 | #MF | CoProcessor Segment Overrun (reserved) | Floating-point instruction. ² |
| 10 | #TS | Invalid TSS | Task switch or TSS access. |
| 11 | #NP | Segment Not Present | Loading segment registers or accessing system segments. |
| 12 | #SS | Stack Segment Fault | Stack operations and SS register loads. |
| 13 | #GP | General Protection | Any memory reference and other protection checks. |
| 14 | #PF | Page Fault | Any memory reference. |
| 15 | | Reserved | |
| 16 | #MF | Floating-Point Error (Math Fault) | Floating-point or WAIT/FWAIT instruction. |
| 17 | #AC | Alignment Check | Any data reference in memory. ³ |
| 18 | #MC | Machine Check | Error codes (if any) and source are model dependent. ⁴ |
| 19 | #XM | SIMD Floating-Point Exception | SIMD Floating-Point Instruction ⁵ |
| 20-31 | | Reserved | |
| 32-255 | | Maskable Interrupts | External interrupt from INTR pin or INT <i>n</i> instruction. |

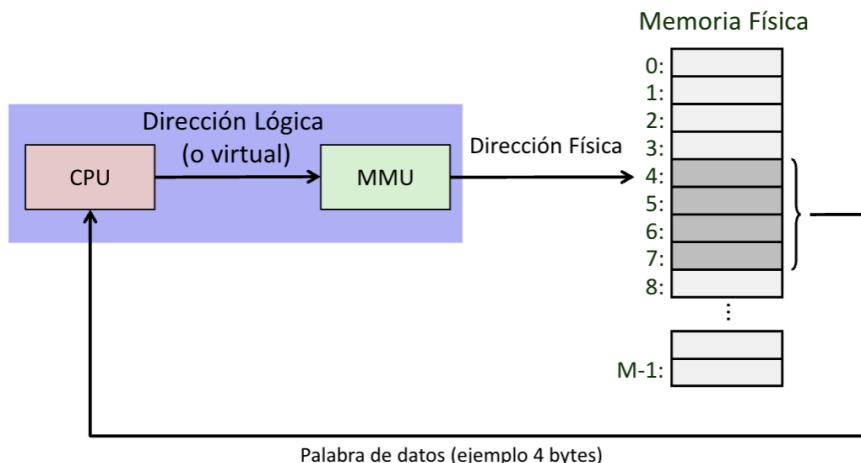
Privilegios de Entrada y Salida

- el procesador provee 2 mecanismos
 - campo de **2 bits IOPL en los EFLAGS**
 - **Nivel 0 (IOPL = 00):** Solo el código que se ejecuta en el nivel de privilegio 0 (*kernel*) puede ejecutar instrucciones de I/O.
 - **Nivel 1 (IOPL = 01):** El código que se ejecuta en los niveles de privilegio 0 y 1 puede ejecutar instrucciones de I/O.
 - **Nivel 2 (IOPL = 10):** El código que se ejecuta en los niveles de privilegio 0, 1 y 2 puede ejecutar instrucciones de I/O.
 - **Nivel 3 (IOPL = 11):** El código que se ejecuta en **cualquier** nivel de privilegio (0, 1, 2 y 3) puede ejecutar instrucciones de I/O.
- Las instrucciones “sensibles” son:
 - IN (Input)
 - OUT (Output)
 - INS (Input String)
 - OUTS (Output String)
 - Cli (Clear Interrupt)
 - Sti (Set Interrupt)
- Para ejecutarlas el nivel de privilegio debe ser igual o menor que IOPL

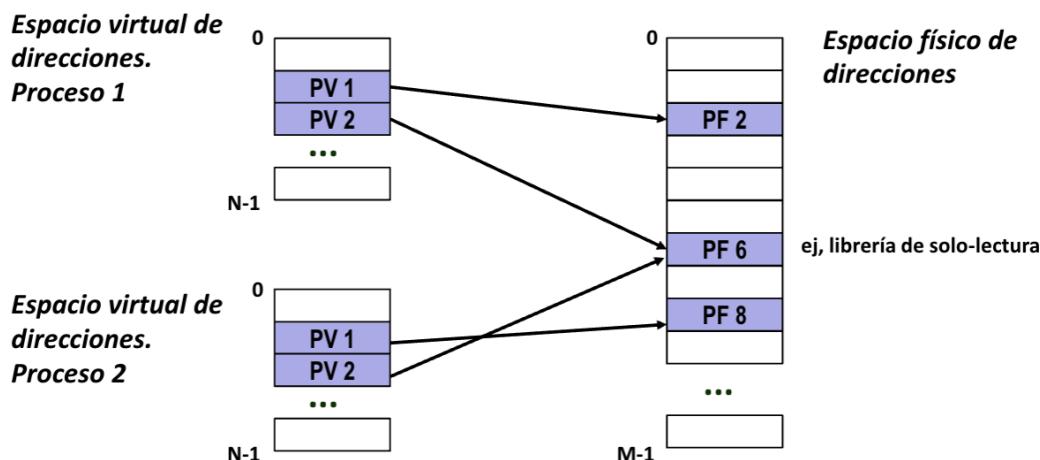
Memoria Virtual.....

Direccionamiento virtual

- Permite asignar direcciones virtuales a los programas, traduciéndose luego a direcciones físicas mediante tablas de paginación.



Memoria virtual para múltiples procesos



- sirve para apuntar muchos procesos al mismo lugar de memoria como cuando usas una librería y así economizar memoria

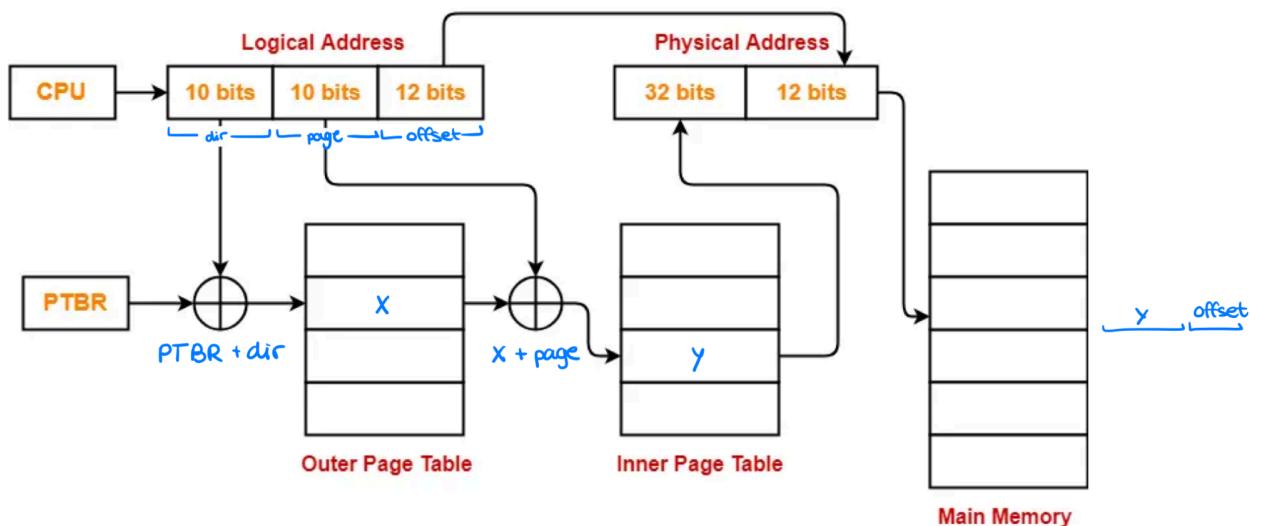
Elección de Intel para 32 bits

| | | | | | |
|-----|----|------|----|--------|---|
| 31 | 22 | 21 | 12 | 11 | 0 |
| DIR | | PAGE | | OFFSET | |

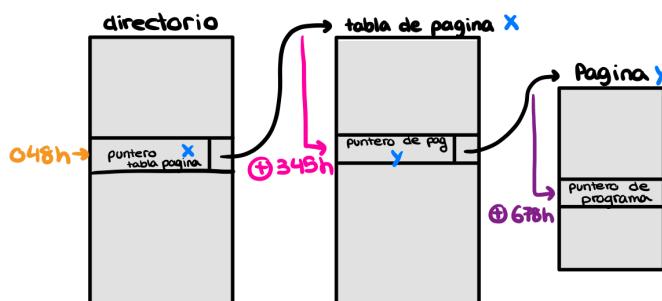
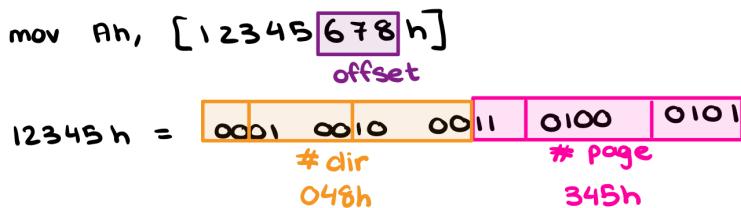
|----- QUE PAGINA ES -----| |----- OFFSET -----|

- los 12 bits menos significativos son offset → dicen en qué posición está dentro de las páginas
- los 20 bits más significativos nos indican qué página es → cantidad de páginas

- 10 más significativos → directorio de página
- 10 menos significativos → página dentro del directorio



ejemplo de mapeo con números



pensar como que directorios son los “capítulos de un libro” y la tabla de página es “número de página en el capítulo”

El problema de la paginación es que puede sobrar espacio, para achicar esto se usa el tamaño medio que necesitan los procesos para el tamaño de página.

ejemplo usando dos páginas para un programa

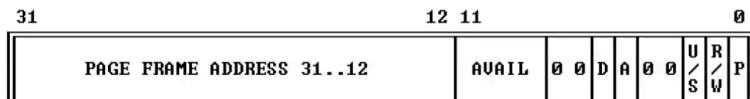
me dan dos páginas → 1234 5000h - 1234 5FFFh
1234 6000h - 1234 6FFFh

la página del ejemplo anterior entra en pagina 1

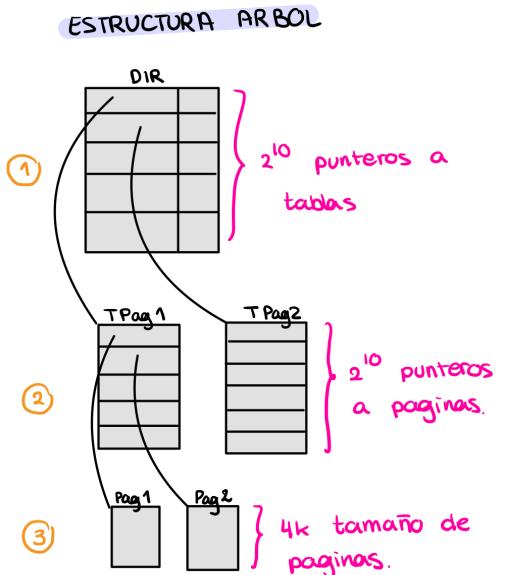
en el MAPA VIRTUAL van continuas PERO en MEMORIA FÍSICA pueden NO estar continuas

Paginación

- el directorio de páginas, pueden referencias 1024 (2^{10}) Tablas de páginas
- luego de cada uno de esas 1024 Tablas de Páginas puede referencias 1024 Páginas, en la memoria FÍSICA
- los elementos de estas tablas tienen el mismo formato como son los elementos dentro de las tablas

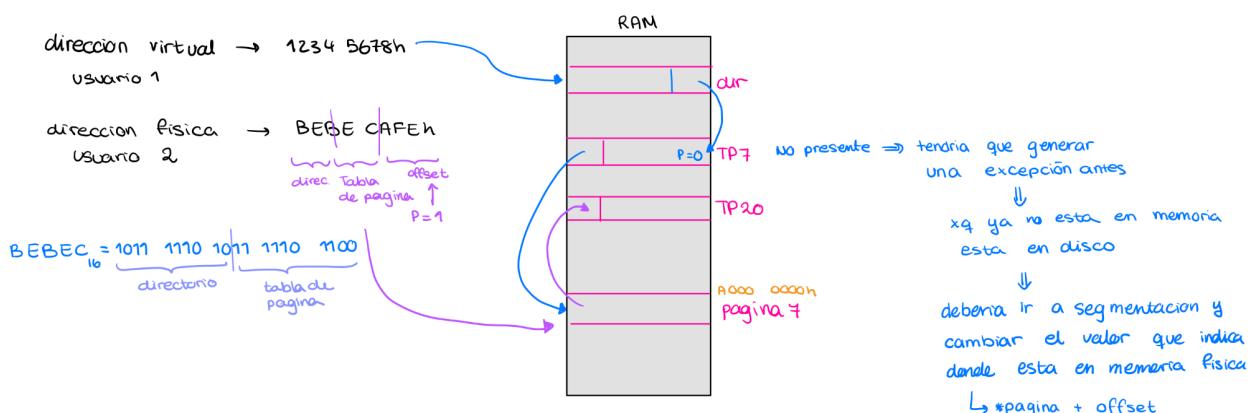


- **P** es el más importante te indica si está presente en memoria o si se fue a disco
- **Page frame address** → es el puntero que te indica la página 4k (caso de tabla de página) o indica la tabla de página (caso de directorio)
- avail → available
- D dirty sí fue modificado
- u/s nivel de privilegio (user/supervisor)
- r/w permisos read/ write



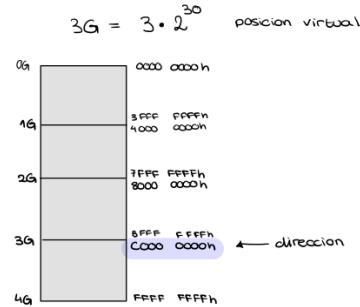
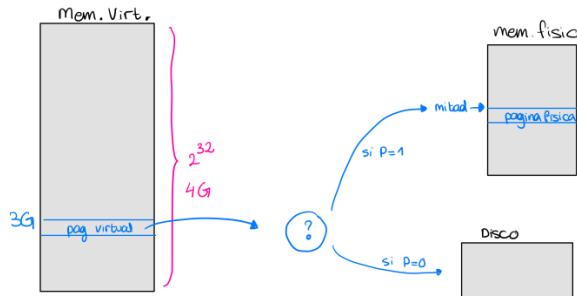
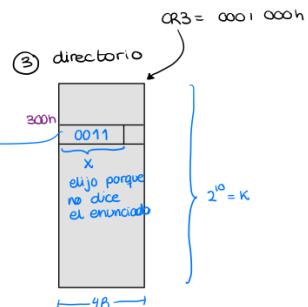
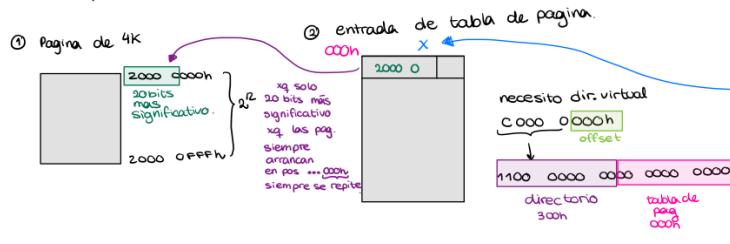
| | |
|----------------------|---------|
| tamaño de una página | 2^n |
| bits de direcciones | m |
| offset de la página | n |
| índice | $m - n$ |

ejemplo con dos usuarios a uno le dan dirección virtual y al otro uno físico y ambos apuntan al mismo lugar físico PERO usuario un tiene una dirección virtual con el bit P = 0

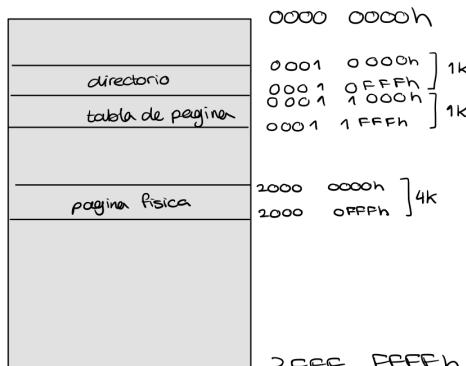


Procesador de 32 bits 1GB de RAM

Quiero ubicar un programa de dirección virtual 3GB pero que corra en la mitad de la memoria física

busco 2000 0000h (dir. virtual) \rightarrow 2000 0000h (dir. física)de atras para adelante \rightarrow 2000 0000h

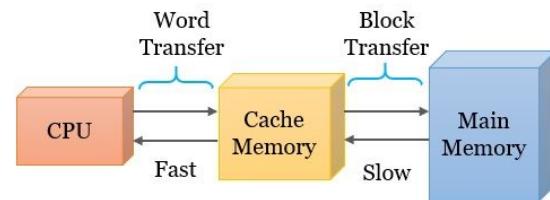
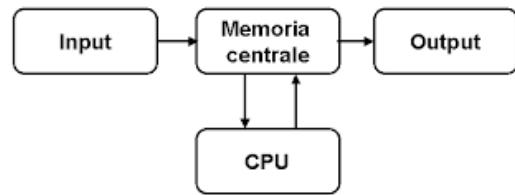
memoria física



Memoria Caché.....

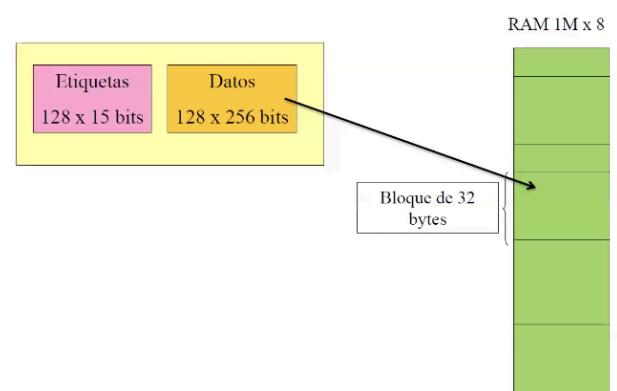
Memoria Caché

- es memoria transparente osea todo lo que programamos no es afectado por lo que hace la memoria caché
- los programas se ejecutan en pasos secuenciales
- las variables se alojan en zonas adyacentes
- **el procesador no habla más con la memoria RAM**
- la memoria caché ve a la RAM en bloques de tamaño fijo
- usa **SRAM (Static)** → a su alta velocidad, baja latencia y capacidad de ofrecer un rendimiento consistente, lo cual es esencial para mejorar la eficiencia del procesador.



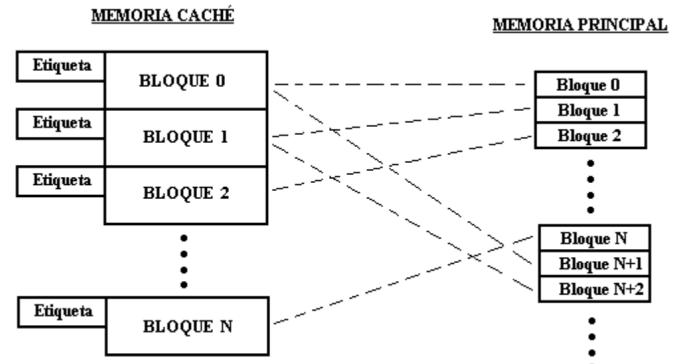
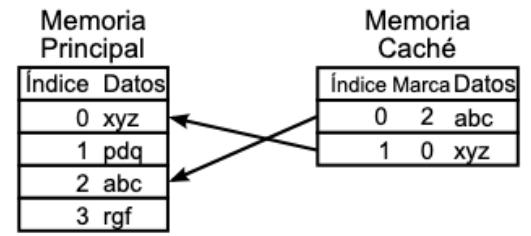
ejemplo:

- El controlador ve a la RAM en bloques de tamaño fijo 32 bytes
- caché de 4k para datos (sin etiquetas)
 - tenemos que dividir la memoria RAM por tamaño de bloques → $\frac{2^{20}}{2^5} = 2^{15} = 32,768$ bloques de memoria física
 - SON etiquetas de 15 bits porque 2^{15} son las posibles combinaciones de los 15 bits de punteros a bloques de datos del caché
 - tenemos que dividir la cache por tamaño de bloques → $\frac{2^{12}}{2^5} = 2^7 = 128$
 - memoria de datos por eso cada entrada tiene 256 bits
 - $8 \text{ bits} * 32 = 256 \text{ bits}$
 - etiquetas es lo mismo que paginación
- **OBJETIVO** de la caché es guardar los últimos datos que venís accediendo que están en la memoria física
- memoria virtual ≠ memoria física



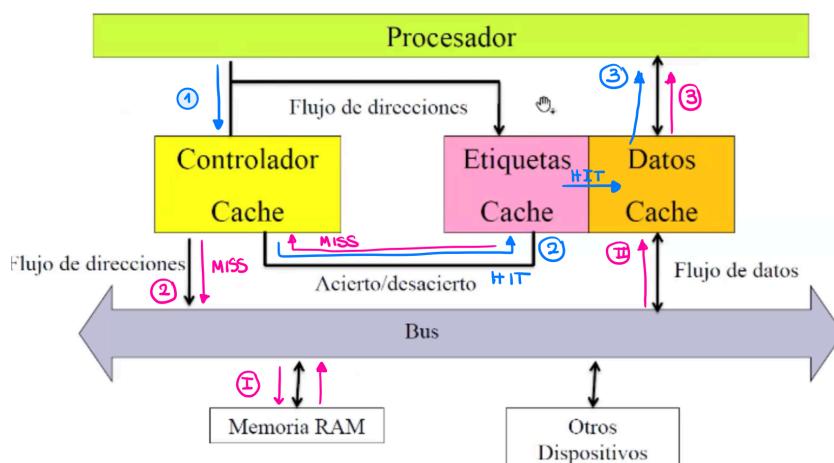
Etiquetas y bloques

- las **etiquetas** son por cada bloque que vos te guardaste en memoria caché tenes que tener un puntero a que bloque es de los 2^{15} bloques posibles (ejemplo anterior)
- Los **bloques** son la unidad mínima de datos que se transfiere entre la memoria principal (RAM) y la caché.
 - contiene no solo los datos solicitados por la CPU, sino también datos adyacentes, con el objetivo de aprovechar la localidad espacial, que es la tendencia de las aplicaciones a acceder a datos cercanos a los que se han accedido recientemente.



¿Cómo funciona?

1. El procesador se fija si tiene el bloque que tiene el puntero del bloque que estas intentando acceder
2. Chequea las etiquetas
 - a. **Hit (Acierto):** Si las etiquetas coinciden, significa que el dato está en la caché.
 - b. **Miss (Fallo):** Si las etiquetas no coinciden, el dato no está en la caché.
 - i. El bloque de memoria que contiene el dato solicitado se carga desde la memoria principal a la caché.
 - ii. Si la caché está llena, se utiliza un algoritmo de reemplazo (como LRU, FIFO, LFU) para decidir qué línea de caché será reemplazada.
 - iii. La etiqueta y el bloque de datos recién cargados se almacenan en la línea de caché correspondiente.
3. El dato solicitado se entrega a la CPU, ya sea desde la caché (en caso de un hit) o desde la memoria principal (en caso de un miss)

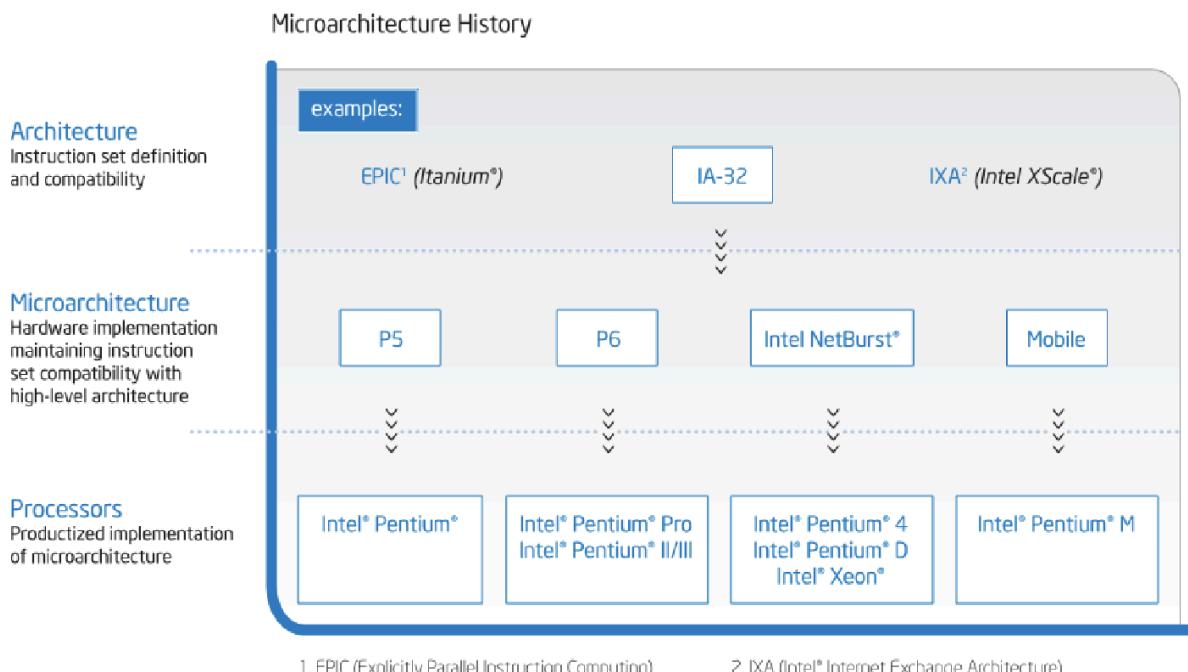


ejemplo:

| | |
|--------|---|
| 33445h | 0011 0011 0100 0100 0101b |
| | 0011 0011 0100 0100 → se busca entre las 128 etiquetas si una tiene el contenido 19A2h = 6562d bloque |

Procesadores de 64 bits

- IA-32 IA-64 AMD64
 - **IA-32:** Tecnología Intel del 32 bits
 - **Intel 64/EM64T:** Extensión Intel de 64 bits, **compatible con 32 bits**
 - **AMD64:** Tecnología AMD de extensión 64 bits, **compatible con 32 bits**.
 - **IA-64:** Tecnología Intel de 64 bits (Itanium) **no compatible con 32.** → FALLO



- El set de instrucciones define la Arquitectura, por lo tanto define la compatibilidad
 - IA-32: Para los Pentium, Celeron, Core Duo, Xeon y Core 2 Duo, etc
 - Intel Xscale: Para los tipo ARM
 - IA-64: Es el set de instrucciones para los Itanium (high end)
- Micro-arquitectura
 - Define como se implementa el hardware
 - Nuevas tecnologías
 - Multi-nucleos
 - Consumo de energía
 - Virtualización
 - Cache
 - FSB
 - Pipelines

ARM.....

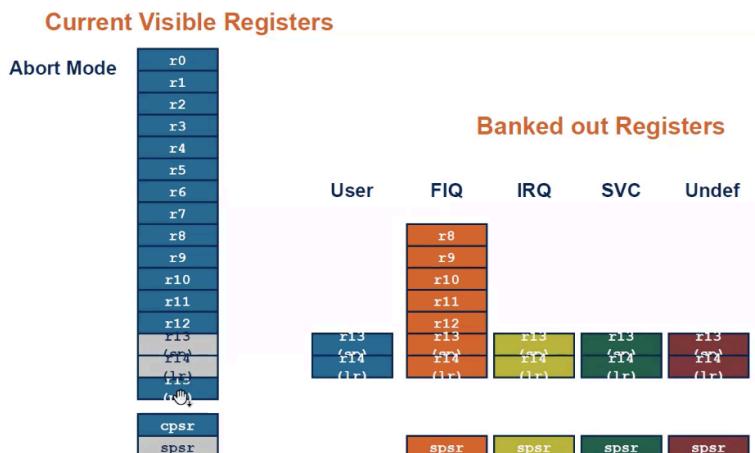
Arquitectura y familias

- ARM7TDMI → arquitectura v4T Von Neumann core con 3 etapas de pipeline
- ARM920T → arquitectura v4T Harvard core con 5 etapas de pipeline

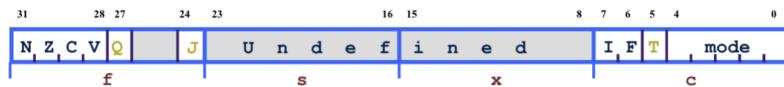
Modos:

| Mode | Description | |
|------------------|--|-------------------|
| Supervisor (SVC) | Entered on reset and when a Software Interrupt instruction (SWI) is executed | Privileged modes |
| FIQ | Entered when a high priority (fast) interrupt is raised | |
| IRQ | Entered when a low priority (normal) interrupt is raised | |
| Abort | Used to handle memory access violations | |
| Undef | Used to handle undefined instructions | |
| System | Privileged mode using the same registers as User mode | |
| User | Mode under which most Applications / OS tasks run | Unprivileged mode |

Set de registros:



- r15 → IP
- 37 registros
- Características:
 - Casi todas las instrucciones se ejecutan en un ciclo de clock y tienen tamaño fijo.
 - Todas las familias de procesadores comparten el mismo conjunto de instrucciones
 - Tipo de datos de 8/16/32 bits.
 - Pocos modos de direccionamiento
 - No se crea fragmentación de memoria



- **Condition code flags**
 - N = Negative result from ALU
 - Z = Zero result from ALU
 - C = ALU operation Carried out
 - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
 - Architecture 5TE/J only
 - Indicates if saturation has occurred
- **J bit**
 - Architecture 5TEJ only
 - J = 1: Processor in Jazelle state
- **Interrupt Disable bits.**
 - I = 1: Disables the IRQ.
 - F = 1: Disables the FIQ. (Fast IRQ)
- **T Bit**
 - Architecture xT only
 - T = 0: Processor in ARM state
 - T = 1: Processor in Thumb state
- **Mode bits**
 - Specify the processor mode

Pipeline en ARM7

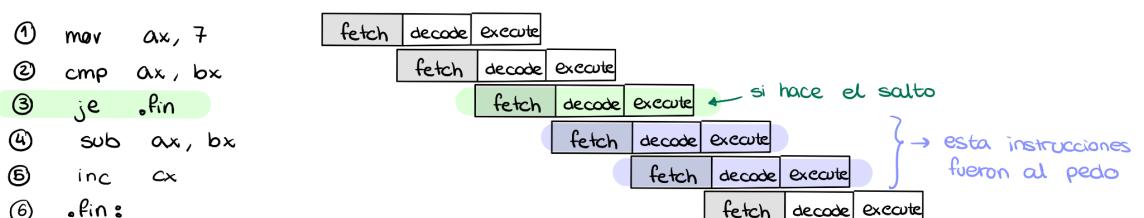
- tiene 3 etapas
 1. **Fetch (Buscar):** En esta etapa, se lee la instrucción desde la memoria y se carga en el registro de instrucciones. El contador de programa (PC) se utiliza para señalar la próxima instrucción a buscar. Mientras se está decodificando y ejecutando la instrucción actual, se pre busca la siguiente instrucción para mantener el pipeline lleno.
 2. **Decode (Decodificar):** En esta etapa, la instrucción buscada se interpreta. Se determina el tipo de instrucción, los operandos que utiliza y cualquier acción necesaria para prepararla para la ejecución.
 3. **Execute (Ejecutar):** En esta etapa, se realiza la operación especificada por la instrucción. Puede implicar cálculos aritméticos o lógicos, lectura o escritura de datos en memoria o cambios en el valor del contador de programa para implementar un salto o una rama.

| | Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 |
|---------------|---------|---------|---------|---------|---------|
| Instruction 1 | Fetch | Decode | Execute | | |
| Instruction 2 | | Fetch | Decode | Execute | |
| Instruction 3 | | | Fetch | Decode | Execute |

Fetch → tarda más que el resto de los pasos

mov por ejemplo tarda en la ejecución porque trabaja con la memoria

- Es adelantarse para ahorrar tiempo PERO cuando hay un jmp genera un problema porque puede ser que haya hecho trabajo sin sentido



LOS Jmps SON PROBLEMATICOs.

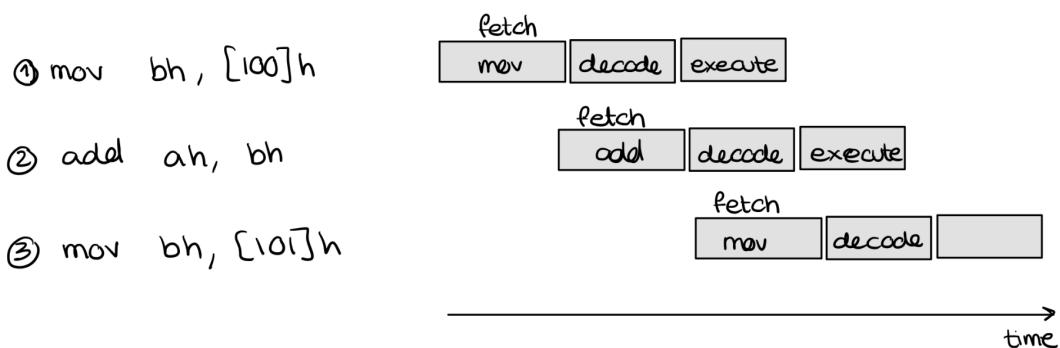
- **SOLUCIÓN** → cuando hay un jmp se ejecuta jmp prediction.
 - intentan predecir si se tomará un salto antes de conocerlo definitivamente, permitiendo así que la pipeline se llene con instrucciones especulativas que pueden ser descartadas si la predicción resulta ser incorrecta.
- instrucciones condicionales
 - le agrego condición a todo
 - se evitan los saltos en el código que demoran la ejecución del programa
 - los 4 bits más significativos son el condicional si se ejecuta o no la instrucción

| | |
|-----------|--------------|
| condición | otros campos |
|-----------|--------------|
- cuando hace el fetch se fija los primeros 4 bits si se cumple
 - si cumple se hace el decode sino se salta y se sigue con la próxima instrucción

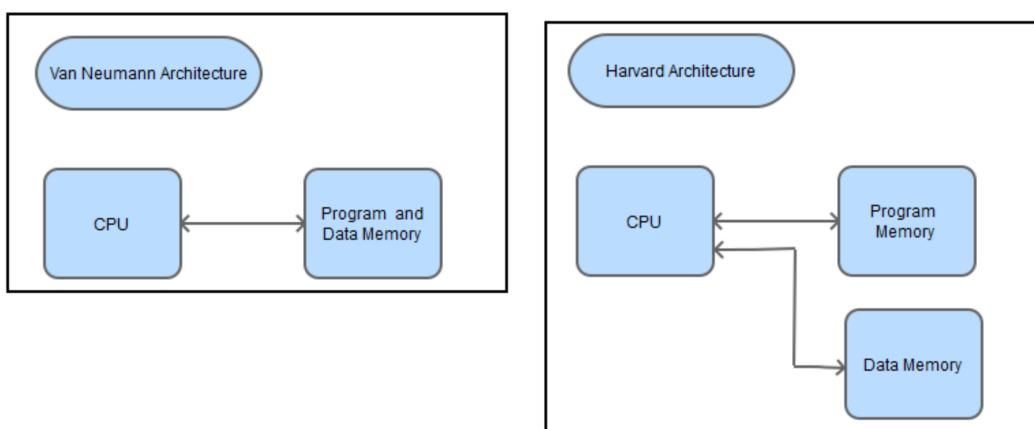
| Suffix | Description |
|--------|-------------------------|
| EQ | Equal |
| NE | Not equal |
| CS/HS | Unsigned higher or same |
| CC/LO | Unsigned lower |
| MI | Minus |
| PL | Positive or Zero |
| VS | Overflow |
| VC | No overflow |
| HI | Unsigned higher |
| LS | Unsigned lower or same |
| GE | Greater or equal |
| LT | Less than |
| GT | Greater than |
| LE | Less than or equal |
| AL | Always |

Arquitecturas Vonn Neumann y Harvard

- supongamos que nuestras instrucciones son:



- cuando lleguemos al ciclo 3 osea cuando se hace el fetch de I3 y execute de I1 como estamos usando arquitectura de **Vonn Neumann** no se pueden hacer las dos en simultáneo ya que ambos usan el bus de datos y por la arquitectura esto **no es posible ya que solo puede ser utilizado por una operación a la vez**.
- **SOLUCIÓN:** Cambiar a una arquitectura **Harvard**, donde las instrucciones y los datos tienen buses separados

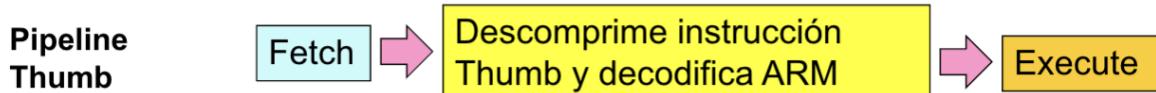


TDMI

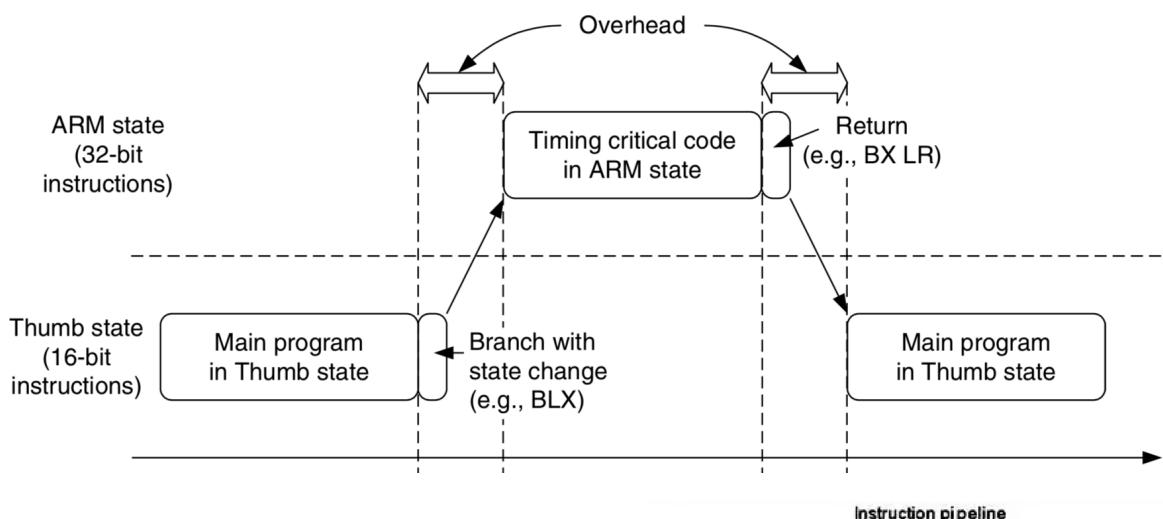
- T: Thumb → set de instrucciones
- D: Debug Interface (JTAG) → permite un HALT del micro para debug
- M: Multiplicador → componente interno del microprocesador, logra resultados de 64 bits
- I: Interrupt → Interrupciones rápidas

Thumb

- microprocesador tiene 2 set de instrucciones
- el ARM clásico de 32 bits y Thumb de 16 bits
- Se descomprimen en forma dinámica en el modulo de “decode” del pipeline

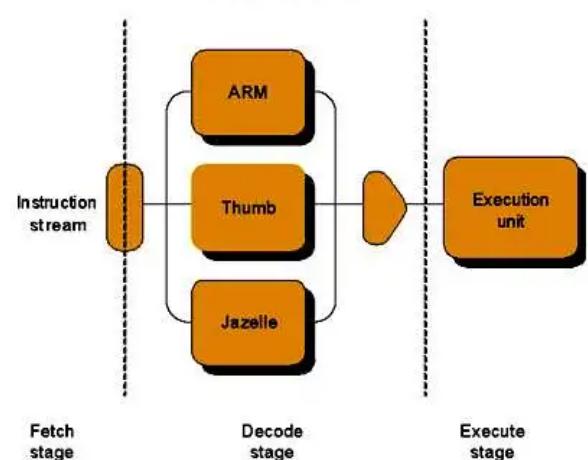


- **BENEFICIO:** Se gana entre un 35% y un 40% de memoria comparado con el set de 32 bits
 - A funciones, por ejemplo, de manejo de caracteres, es mejor Thumb
- Se puede switchear de Thumb a ARM en forma dinámica, para aplicaciones combinadas (16 y 32 bits) y así no perder performance.



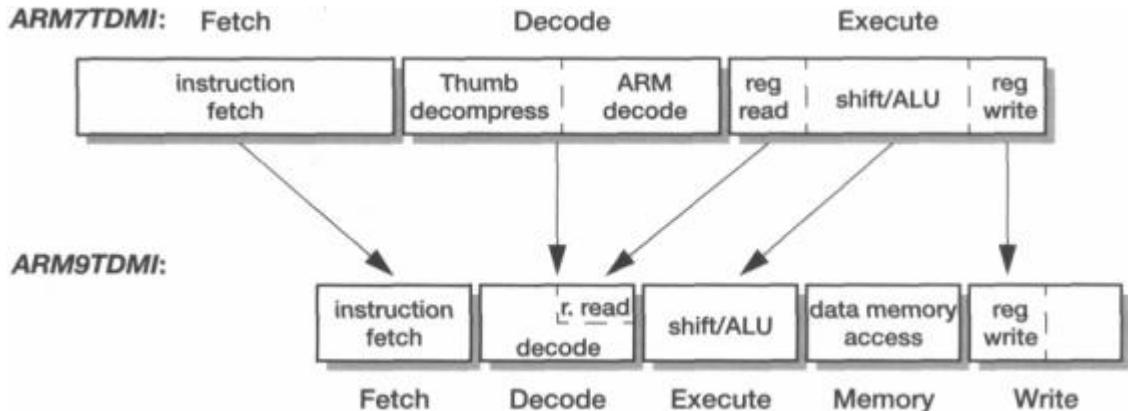
Jazelle

- los procesadores con extensión Jazelle, son capaces de **ejecutar bytes code de Java** directamente en hardware
- las instrucciones de Java que **no posee las emula con las instrucciones ARM**



ARM9TDMI

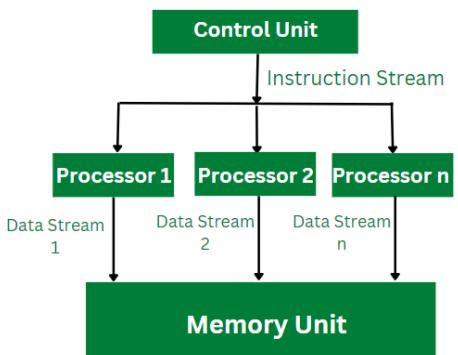
- es reemplazo de los ARM7
- es compatible a nivel binario con ARM7
- tiene un pipeline de 5 niveles en lugar de 3 → permite aumentar la frecuencia del clock
- con memoria caché
- arquitectura de Harvard modificada. Instrucciones y datos separados en caché pero mismo espacio de memoria → mismo mapa de memoria como vimos arriba



GPU (Unidades de procesamiento gráfico).....

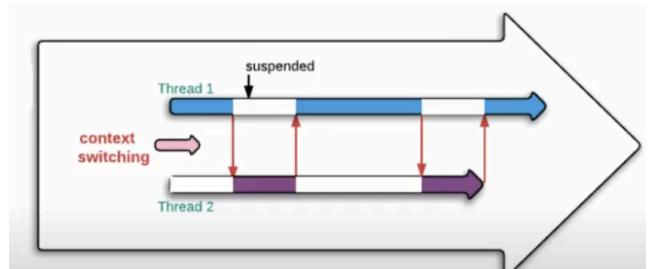
Computación Paralela

- división de tareas de cómputo
- puede encontrarse en diferentes niveles
 - pipeline
 - threads
 - multi-cores
 - APIs → no APIs de desarrollo de software
 - Clusters - Grids



Threads

- parte de un programa que se puede ejecutar de manera independiente de otras partes de código
- se implementan junto al SO con una librería
- un proceso puede tener por ejemplo 2 hilos
- **corren dentro del mismo núcleo (core)**
- al compartir recursos
 - ej. memoria se debe cuidar la sincronización
 - ej. exclusión tipo **mutex** (mutual exclusion) → para usar mismos recursos por ambos hilos



| Fork: | Thread: |
|--|--|
| crea un “child process” → NO hace un hilo SINO un proceso entero | crea un “sibling process” → crea una parte de ese proceso en paralelo |

GPU

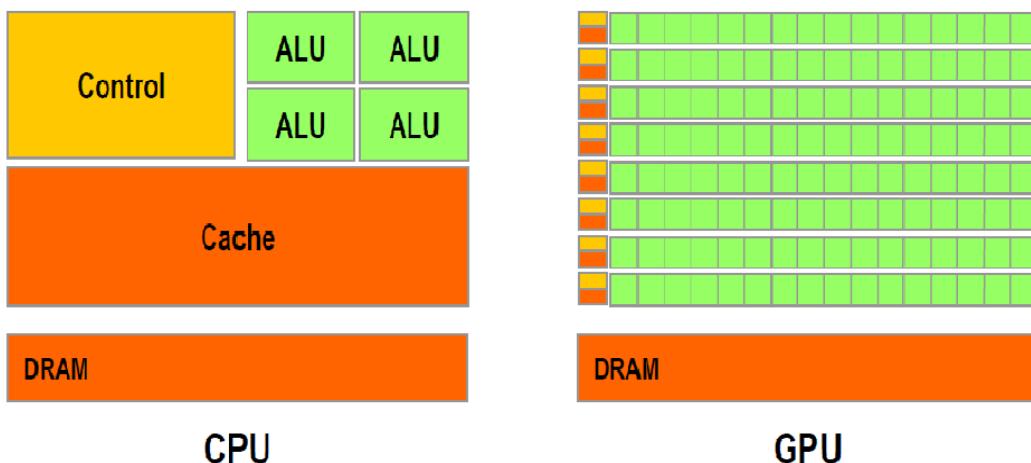
- diseñada para ejecutar miles de threads en paralelo
- menor velocidad de ejecución de un thread que una CPU
- pero más cantidad
- **GPU < CPU velocidad**

CUDA

- es un pipeline de operaciones de punto flotante
- se programa en C o C++

Paralelización

- en CPU de un hilo, recorre cada elemento, sumo e incrementa, de manera secuencial
- en CPU de dos hilos de podría paralelizar, sumar elementos a la vez
- EN CAMBIO GPU cada suma podría ser ejecutada simultáneamente por un procesador (virtual) de la GPU



Pasos para ejecutar código en GPU

1. Se copia los datos de entrada de host-memory a device-memory
2. Se carga del programa se copia de device-memory y se ejecuta
3. la salida del programa se copia de device-memory a host-memory

HOST → CPU

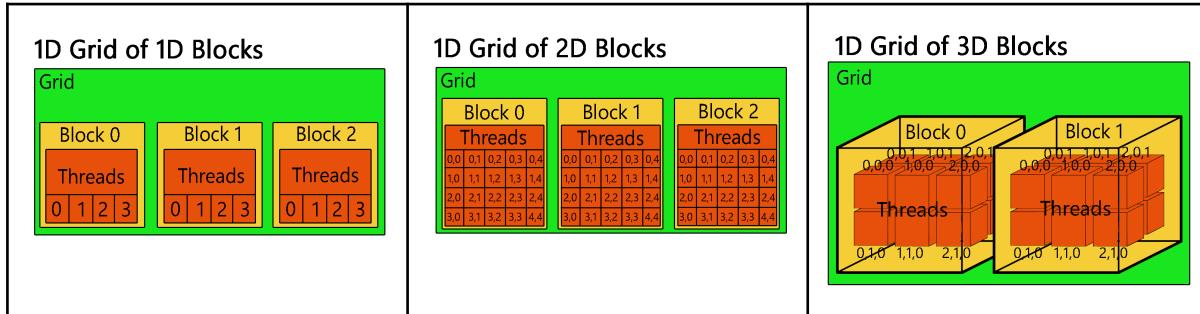
DEVICE → GPU

Hay que agrupar los hilos

- Block → grupo de hilos → como la tabla de páginas
- Grid → grupo de blocks → como la tabla de directorios
- SM (Stream Multiprocessor)
 - puede correr N blocks
 - Cada GPU contiene diferentes cantidades de SM

Grids y Blocks

- Bloques con hilos de una, dos y tres dimensiones



- Cada hilo sabe a qué bloque corresponde y cuántos bloques hay en el grid

Un programa CUDA al correr en una GPU con 4 SM o en una GPU con 8 SM. Es transparente para el programador. Lo soluciona CUDA

Como lo corre depende de cuánto SM tiene tu procesador

- en CUDA al programar se debe definir cuántos hilos serán usados en cada bloque
 - Para eso existen las variables:
 - blockIdx: Id del bloque
 - threadIdx: Id de hilo dentro del bloque
 - blockDim: Número de hilos por bloque
 - estas variables tienen un atributo por eje cartesiano x,y,z

Machine Learning

- en redes neuronales los nodos pueden realizar su tarea computacional de manera paralela