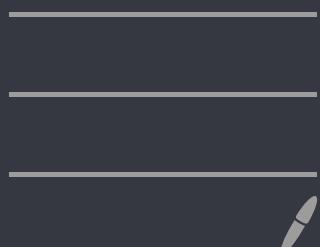


# ARQUI FINAL



★ Significa que se preguntó en un final

LEER NOTION  
IVAN CHAYER



# ASM y C

Instrucción de ASM

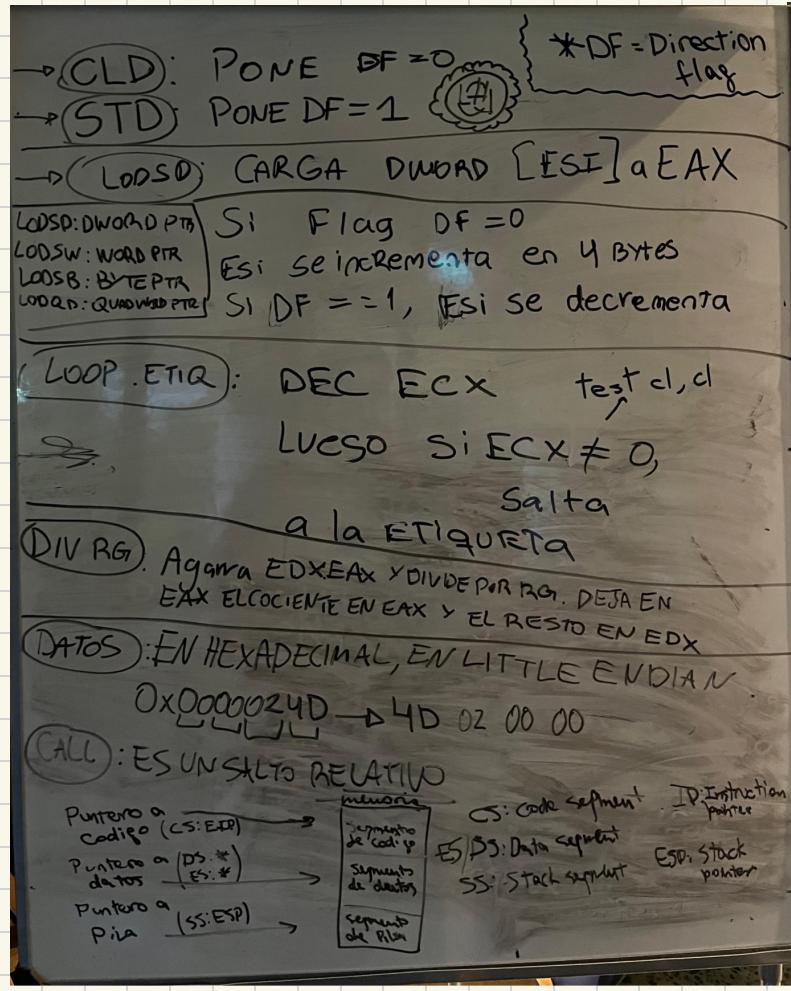
• ret: hace un pop y JMP a la dirección que estaba en el tope del stack

> Ret vs exit

syscall del SO

• exit: propia de C, varios pasos, desarmado de stack-frame. Termina con un ret

► Stack-chk-fail

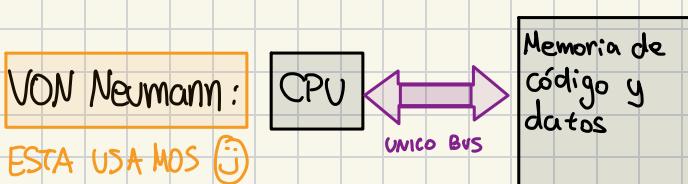
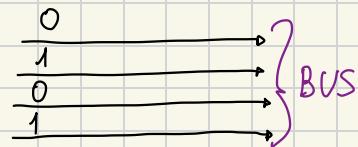


nasm -f elf32 file.asm -o file.o  
 ld -m elf\_i386 file.o -o program  
 gcc -m32 archivo.c fileASM.o -o program  
 gcc -c → genera código objeto (.o) NO EJECUTABLE  
 LEAVE, RET == MOV ESP, EBP; POP EBP; POP EIP  
 EBP AM RET AM  
 EG. a PRESERVAR: EBX, ESI, EDI, EDX, ESP  
 ALIGN 4  
 SIEMPRE  
 0 0 0 0  
 0 H 0 0 0 0  
 0 0 0 0 0 0  
 0 0 0 0 0 0

EN ASM:  
 BYTE: 1 byte  
 WORD: 2 bytes  
 DWORD: 4 bytes  
 QWORD: 8 bytes

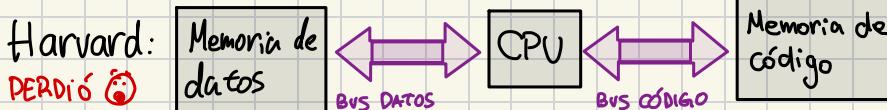
# ARQ. DE LAS COMPUS

Transmisión: en Serie: 0101 → un solo canal VS. en paralelo:



2 ciclos para ejecutar una instrucción:  
Fetch del código y luego fetch del dato necesario.

Única dirección física para datos y códigos

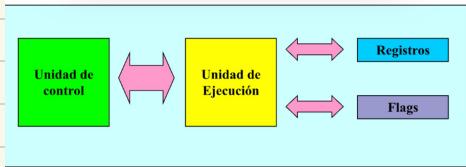


1 ciclo x instrucción: puede hacer fetch del código y de los datos en simultáneo.

2 dir. físicas: para datos y para código

CS: Apunta al comienzo del code segment.

EIP: Offset desde el CS indicando la dirección de la sig. instrucción a ejecutar.



Existen 2 arquitecturas de 64 bits de Intel.

Existen 2 arquitecturas de 64 bits de Intel: IA-64 y x86\_64.

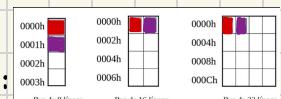


IA-64 Utiliza el conjunto de instrucciones EPIC (Explicitly Parallel Instruction Computing), que es significativamente diferente de la arquitectura x86 tradicional. Esta arquitectura no es compatible de manera nativa con las aplicaciones x86, aunque puede ejecutar aplicaciones x86 mediante emulación.

Por otro lado, x86\_64 tiene un set de instrucciones compatible con el de la arquitectura x86 intel, lo que logra que se puedan ejecutar programas escritos para x86 de manera nativa. Los registros de 64 bits son continuaciones de los de 32 bits, y comparten el set de instrucciones.

## Memoria

Mapa de memoria: tamaño  $2^{B.\text{Address}}$ . Ej: 16 BA, 16 BD:  $2^{16} = 2^6 \cdot 2^{10} = 64k$  y 16 BD: 16 bytes = 2 B (bytes)  
 $64k \cdot 2B = 128 kB$  espacio de memoria



(aunque la memoria se accede con el ancho del BD)

Palabra: En intel, palabra de 1 byte:

IP: Arranca cableado en una dir.; En intel: FFF0h. Inicialmente, se lee código de la ROM, el cual carga el SO a la memoria RAM. (Creando el kernel space en la RAM!)

**ROM** { + lentas + baratas - No volátil (no necesita energía) ► PROM (Programmable); EEPROM (Erasable)  
 • Contienen el código para bootear el SO (Se usan para datos q' no cambian frecuentemente) ► EEPROM (Electric) y Flash, las de hoy en día SD y Pendrives

**RAM** { Volatil (necesita energía p' conservar datos) ► DRAM (Dynamic) + barata + lenta - Compleja A esta nos referimos con "memoria RAM"  
 ++ rápida + cara  
 Se usa para todas las tareas de la compu. ► SRAM (Static) + cara + rápida + Compleja Se usa para la memoria caché  
 - no refresh needed

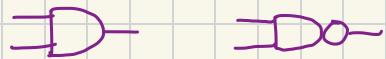
# Decodificación

	$\bar{I}$	$I_1$	$I_2$	$I_1 \cdot I_2$	$(I_1 + I_2)$	$I_1 \cdot \bar{I}_2$	$I_1 + \bar{I}_2$	$\bar{I} \cdot \bar{I}_2$	$\bar{I} \cdot I_2$
1	NOT	1	2	AND	1	2	NAND	1	OR
0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	1	0	1	0	1
-DO-									
		0	1	0	0	1	0	1	1
		1	1	1	1	0	1	0	0
		0	1	0	1	1	0	0	1
		1	1	1	0	1	1	1	0

	$\bar{I}$	$I_1$	$I_2$	$I_1 \cdot I_2$	$I_1 + I_2$	$\bar{I} \cdot \bar{I}_2$	$\bar{I} \cdot I_2$
1	NOT	1	2	AND	1	2	OR
0	1	0	0	0	0	0	0
1	0	1	0	0	1	0	1
-DO-							
		0	0	0	0	1	0
		1	0	1	1	0	1
		0	1	1	1	0	1
		1	1	0	1	1	0

	$\bar{I}$	$I_1$	$I_2$	$I_1 \cdot I_2$	$I_1 + I_2$	$\bar{I} \cdot \bar{I}_2$	$\bar{I} \cdot I_2$
1	NOT	1	2	AND	1	2	OR
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
-DO-							
		0	1	1	1	0	1
		1	1	0	1	0	0
		0	1	0	1	0	1
		1	1	0	1	0	0

	$\bar{I}$	$I_1$	$I_2$	$I_1 \cdot I_2$	$I_1 + I_2$	$\bar{I} \cdot \bar{I}_2$	$\bar{I} \cdot I_2$
1	NOT	1	2	AND	1	2	OR
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
-DO-							
		0	1	1	1	0	1
		1	1	0	1	0	0
		0	1	0	1	0	1
		1	1	0	1	0	0



SOLO INTEL SOLO INTEL!

IO/M: 1 operaciones con pares, 0 con memoria → Mapa de E/S, con igual # de direcciones (y con las mismas dir. numéricas)

Si se agregaba otra pata al BA, se rompía la retroc. y la lógica de pot. de 2

OBS: hoy en día los perif. se mapan directo al mapa memoria (se conserva el mapa E/S, con  $2^{16}$  direcciones) Siempre 16 bits

ventaja: Son más las instrucciones de manejo de memoria. "Desventaja": gasto memoria, pero hoy día sobra!

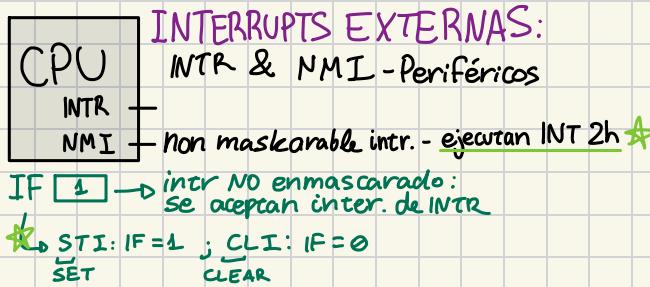
D/WR: Vale 1: hay escritura D/RD: Vale 1: hay lectura

Deco completa: relación biúniciva entre pos. de memoria y dirección (no dejar patitas del BA libres)

Si no se conecta todo el BA o searma mal la deco, aparecen 'imágenes' (2 pos. de memoria acc. por 1 dirección)

DMA (direct memory access): transmisión de datos entre periférico y memoria sin intervención del CPU, por ejemplo disco duro y memoria RAM

## Interrupciones



### INTERRUPTS de SOFTWARE

Se invocan con Assembler "INT xh"

puntero a interrupt

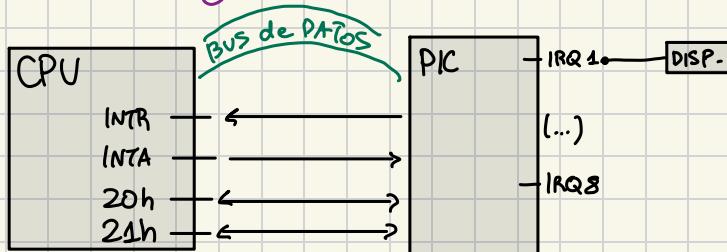
### RUTINA de INTERRUPT

Al generarse una interrupt, el CPU busca el puntero a la rutina, la ejecuta, y luego finaliza con un IRET (restaura registros, flags e IP)

EXCEPTIONS: Interrupciones internas, generadas por el mismo CPU. Al invocarse, el CPU tmb va a la IDT, por lo que el SO (y/o la BIOS) deben agregar punteros a rutinas. Son las primeras 32 entradas de la IDT, y se usan solo 20.

IDT: Está en Memoria. Es llenada por la BIOS y luego pisada por el SO. ES UNICA

### PIC, 2PICs y APIC

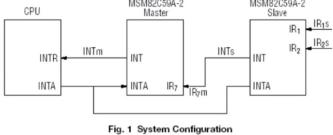


- Los disp. interrumpen al PIC
- El PIC resuelve la cola e interrumpe al CPU por INTR
- Si IF=1, el CPU avisa al PIC que recibió x INTA
- Al recibir INTA, el PIC envía la int. por el BD
- El CPU busca la rutina en la IDT y la ejecuta
- Al finalizar la rutina, el CPU envía por el puerto 20h el EOJ, el valor 20h ... mov AL, 20h out 20h, AL

Máscaras: Por el puerto 21h se programa la IMR, un vector de 8 bits indicando cuales IRQ enmascarar (1:enmascarada)

Programación del PIC en el puerto 20h (prioridades y funcionamiento) - Lo usa la BIOS

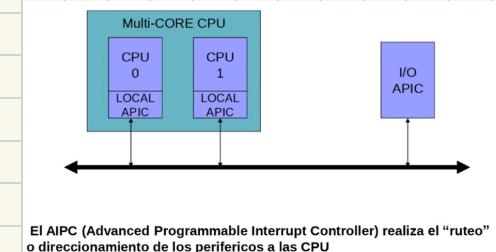
## PIC en cascada



Colocando 2 PICs en cascada se amplia la cantidad de interrupciones de hardware en la PC.

En la PC, se utiliza el IRQ2 del Master para conectar el Slave.

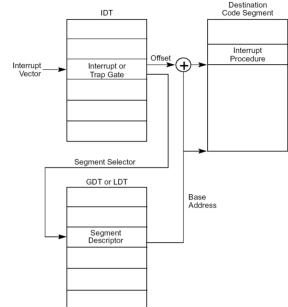
## Multicore: APICs \*



Los periféricos se conectan al LOCAL APIC; Este decide a **Cual core** enviar Cada interrupt. Cada core posee su propio **LOCAL APIC**.

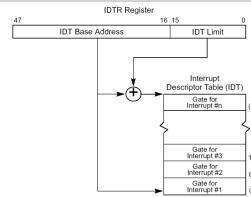
- Para los procesadores multicore, las interrupciones son:
- o a. Individuales en cada procesador
  - o b. Centralizadas por APIC
  - o c. Centralizadas por 2 PIC
  - o d. Ninguna de las anteriores.

\* **TIMER TICK**: Integrado que interrumpe al CPU C/55 ms. Se conecta al Master PIC IRQ0.



### IDT - MODO PROTEGIDO

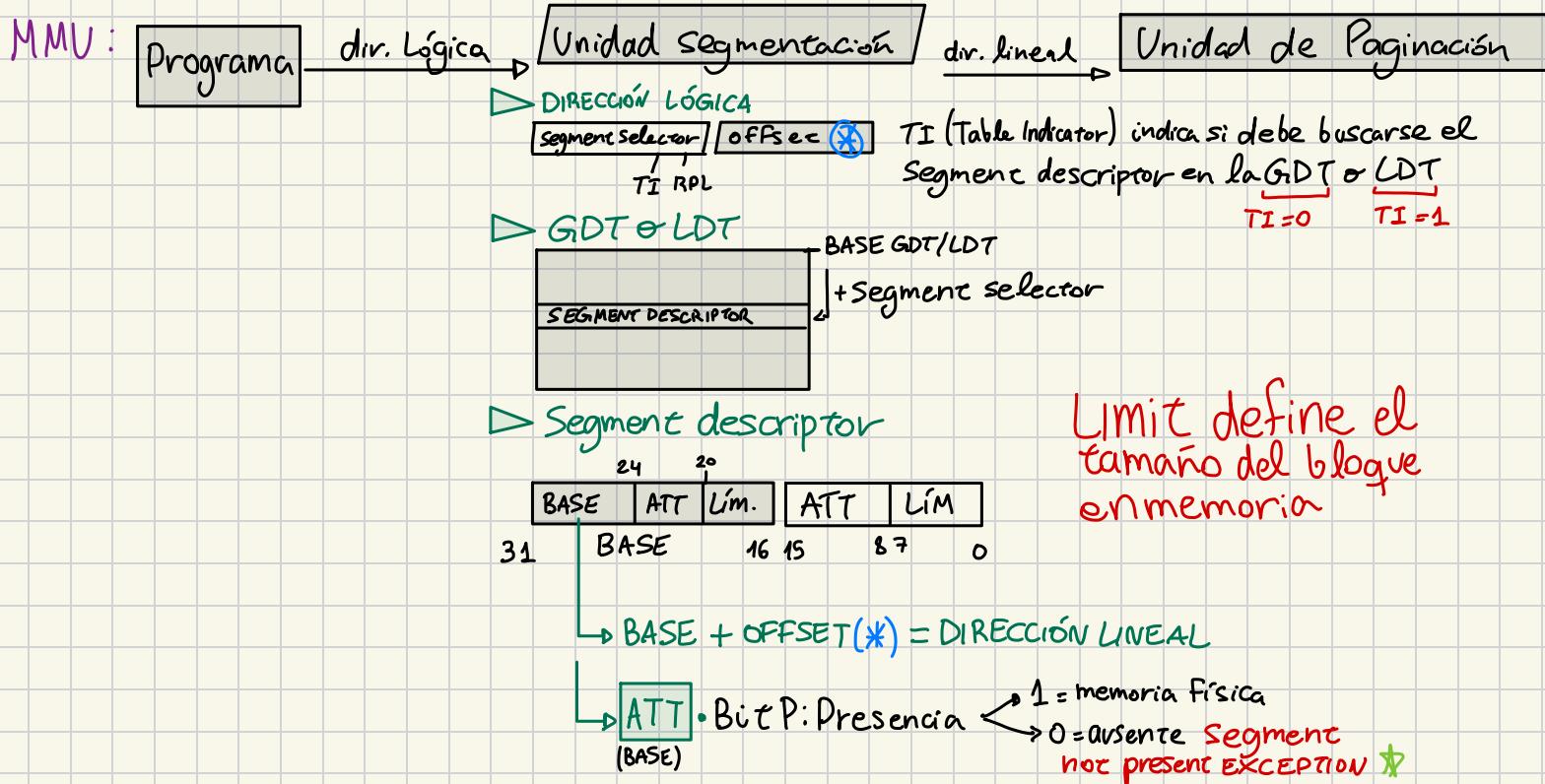
Las entradas de la IDT son punteros a una tabla GDT o LDT, donde el Segment descriptor contiene **los permisos necesarios**, y la base y el offset para hallar el primero red a la vecina.



**REGISTRO IDTR** (en el CPU) contiene la base de la IDT, y el IDT Limite:table size. Lo completa el SO.

## MODO PROTEGIDO: SEGMENTACIÓN

\* **Modo Flat**: se configuran todas las páginas para abarcar todo → Las direcciones el espacio de direcciones, y ser accesible para todos los PL. **Lógicas == Lineales**



\* **DPL**: descriptor permission level, requerido para acceder al segmento

\* **S**: Tipo de Segmento → 1 normal: Código, datos o pila  
↳ 0 SISTEMA (TRAP GATE, TASK GATE, INTERRUPT GATE, etc.)

**PRIVILEGIOS**: PL: Permission Level; Va de 0 (So/kernel) a 3 (user)

\* **E/S**: Dados por los bits IOPL (Input Output PL) en los EFLAGS en el CPU.  
Son los permisos necesarios para ejecutar instrucciones de E/S.

**CPL**: Current Privilege Level, contenido en los registros de CS (code segment).  
Son los permisos del contexto actual, los que chequea el CPU antes de ejecutar instrucciones en modo protegido.

# ARM

Advanced RISC Machine

SoC System on a chip: Perifericos ya integrados (ROM, RAM, Flash, interfaces I/O, etc) en el CPU

Modos ofrece 7 modos. Cada modo tiene su propio set de registros y stack. Cada modo tiene sus propios privilegios. Hay 2 modos 'estandar'; user y supervisor, y 5 de EXCEPCIÓN.

Aprender Jazbel THUMB ecc